Cloud 101

Building Servers as Cattle not Pets using Infrastructure as Code

Part 1
GUI Cloud Instance Creation

Part 2
Infrastructure as Code with Terraform

Skill Level: Beginner - No previous experience with software development or cloud required.

Course Length: 1.5 hours

Prerequisites:

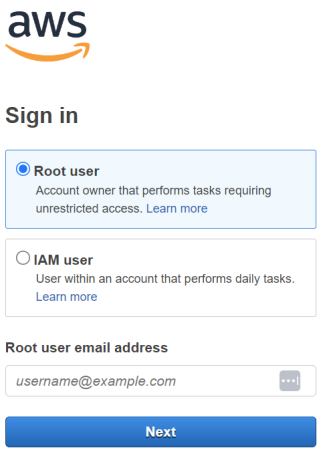
- Install Visual Studio Code
- Download Putty
- Bring your own AWS Account
- Github Repo Access

Presenter: David Thurm
Contact:

- Linkedin: davidthurm
- Twitter: @davidthurm

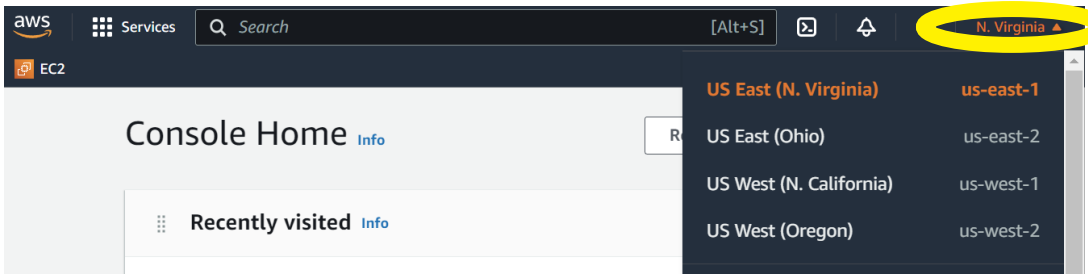
Logging into the AWS Console

Console URL: https://console.aws.amazon.com/
Login Info: The AWS account you setup earlier.



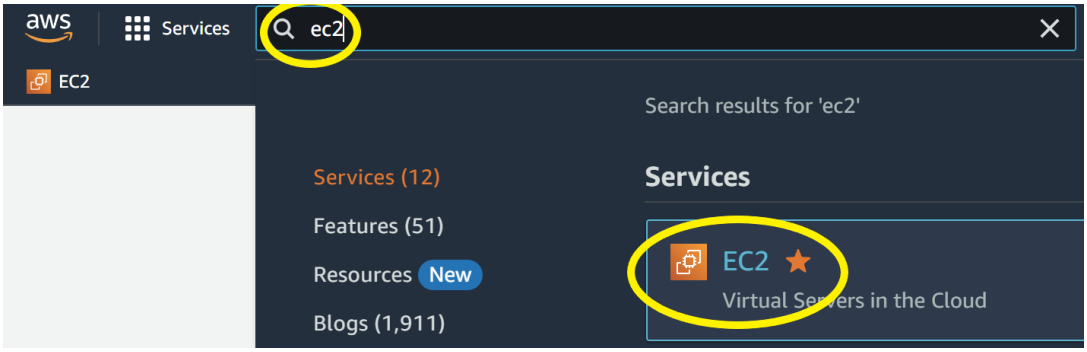
Before We Begin

Step 1: Verify "US East (N. Virginia)" is selected

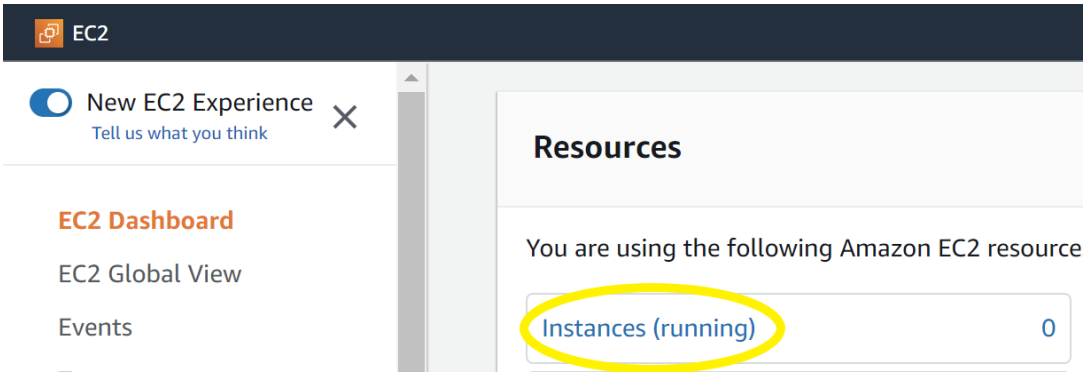


Create a new EC2 Instance

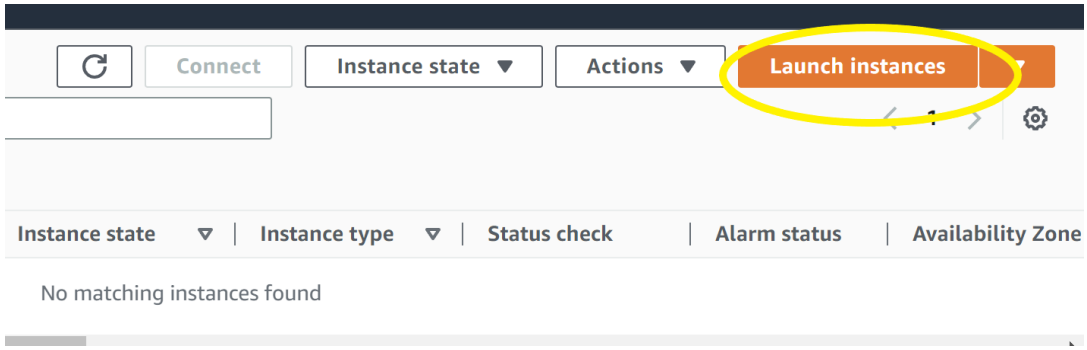
Step 1: Search > **EC2** > Select the EC2 Icon



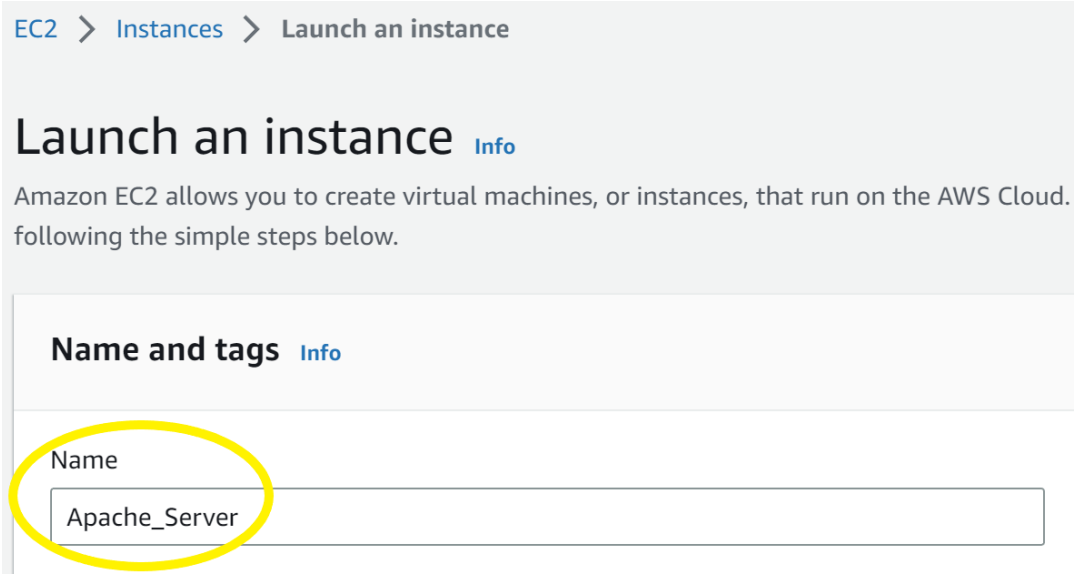
Step 2: Select > Instances (running)



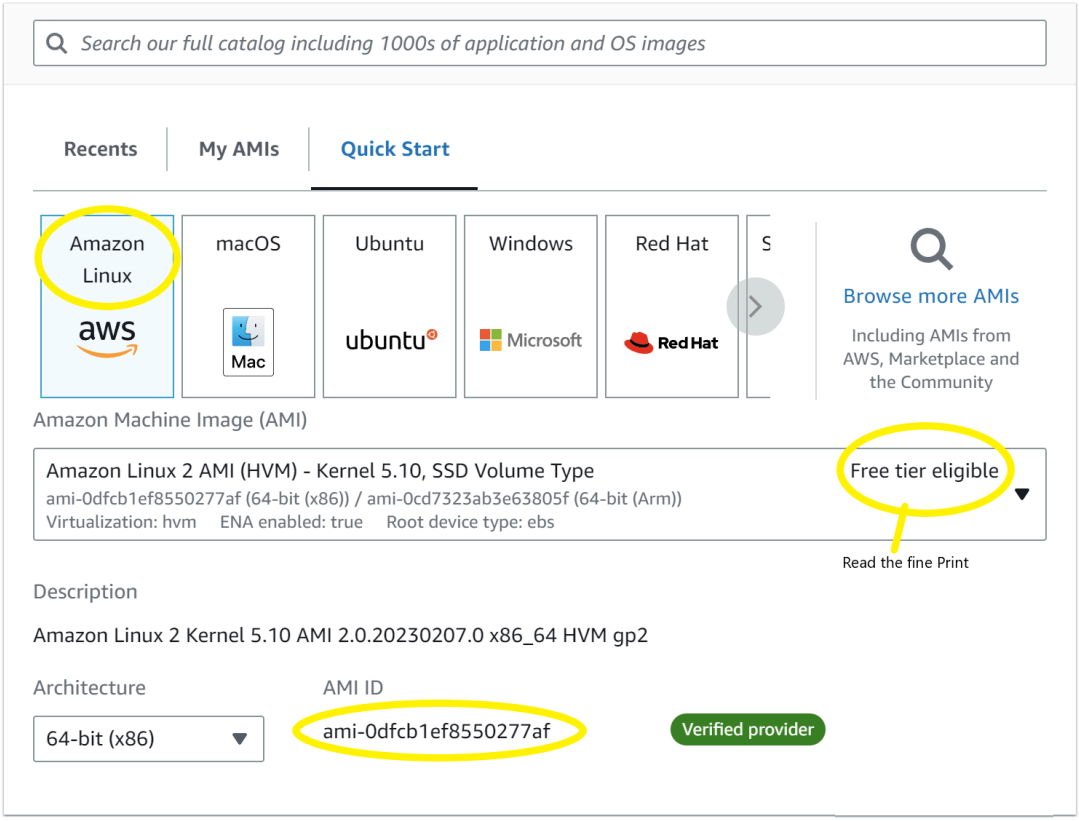
Step 3: Select > Launch Instances



Step 4: Name the Server > "**Apache_Server**"



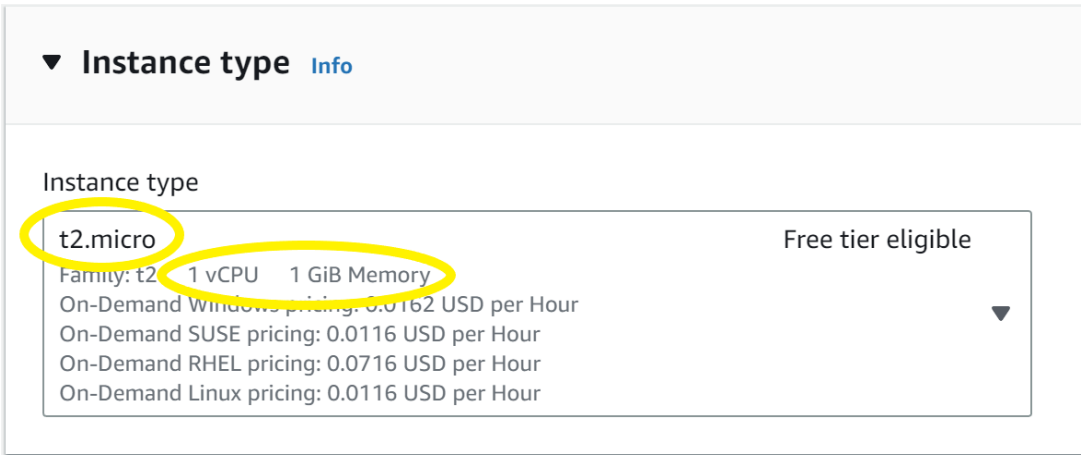
Step 5: Choose your OS (ie Amazon Linux)



Not all operating systems are FREE.

Copy the AMI ID at the bottom to your notepad. We will use it later.

Step 6: Choose “**t2.micro**”



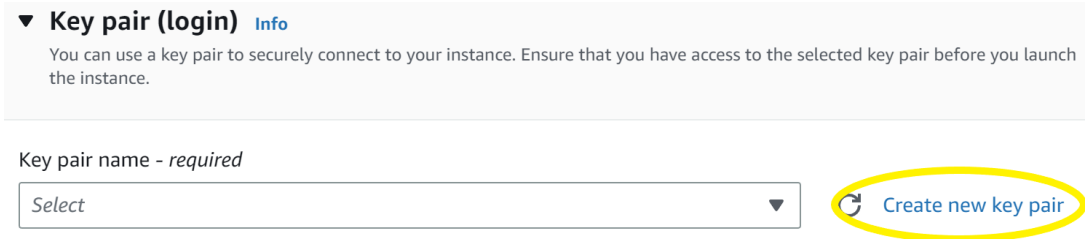
Key details are the CPU and Memory Available.
In this case above we see the server specs are barely has enough to run anything.

Free Tier Conditions:

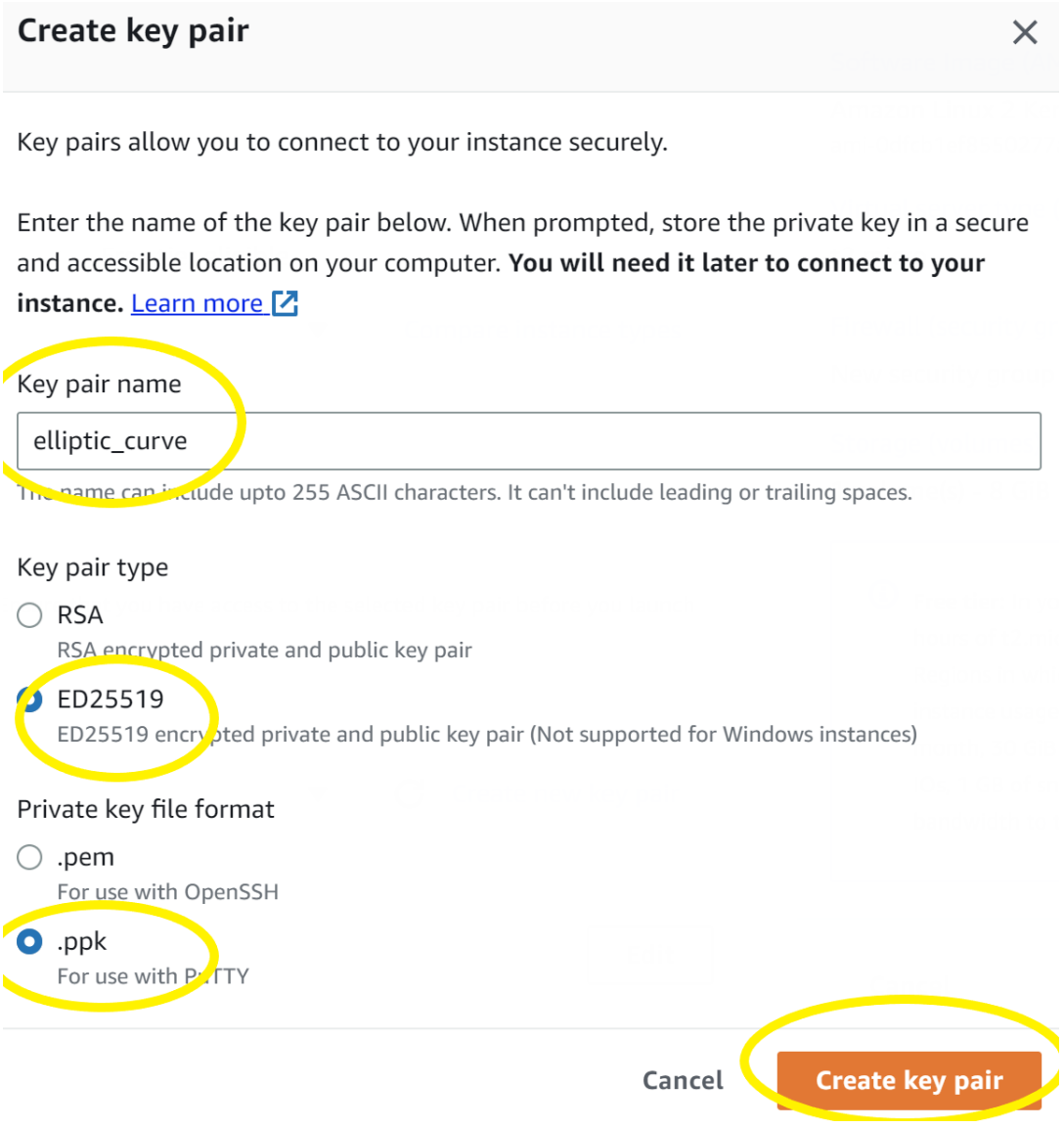
- Operatings Systems bought in AWS Marketplace are **NOT** free.
- Only good for the first 12 months of your initial aws sign-up date.
- Some things have a trial period of less than the 12 months.

Create a New Key Pair

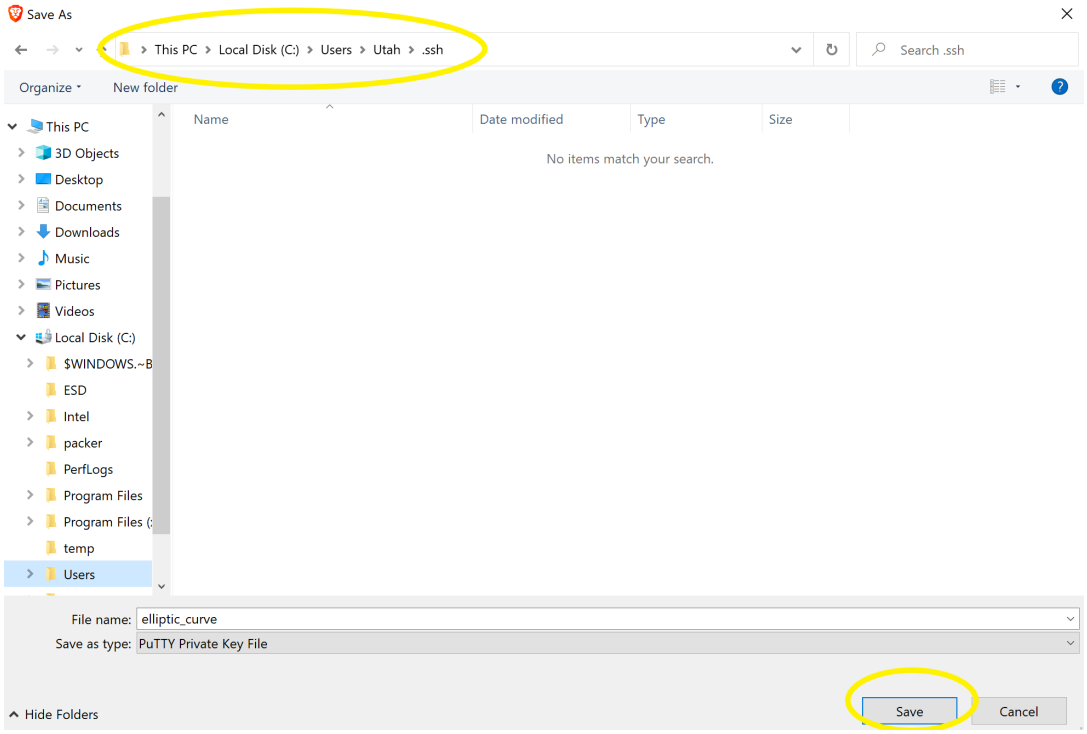
Step 1: Click "**Create new key pair**"



Step 2: Name your key pair > "**Elyptic_Curve**"
Step 3: Choose Elypic Curve "**ED25519**" (Notice there is no Windows support)
Step 4: Choose "**.ppk for use with putty.**"
Step 5: Click "**Create Key Pair**"



Step 6: Change directory to: C:\Users***your-username***\
Step 7: Make a new folder named .ssh inside this folder
Step 8: Change directory to: C:\Users***your-username***\.ssh
Step 9: Click “Save”

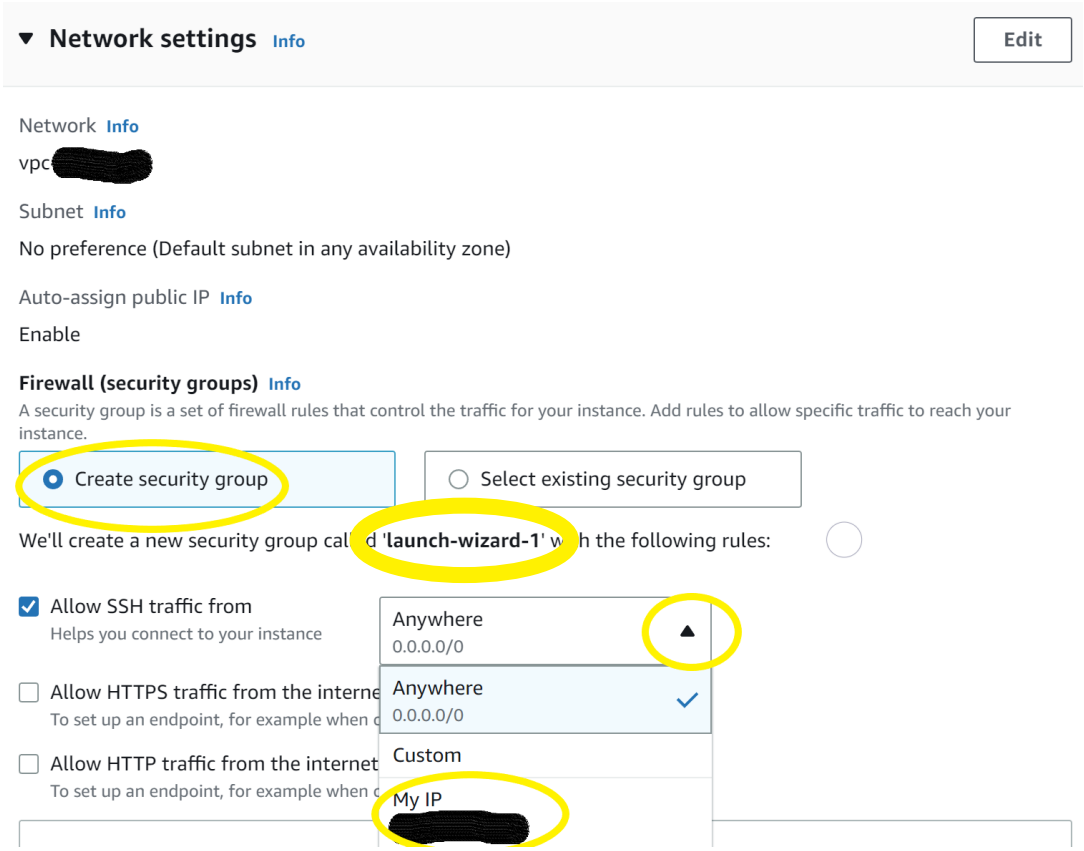


The use of the folder we crated above is to provide a standardized location for all your ssh keys. As time goes on you will have several for different purposes.

We may need to move the file from the downloads to the folder we created.

Create a Security Group

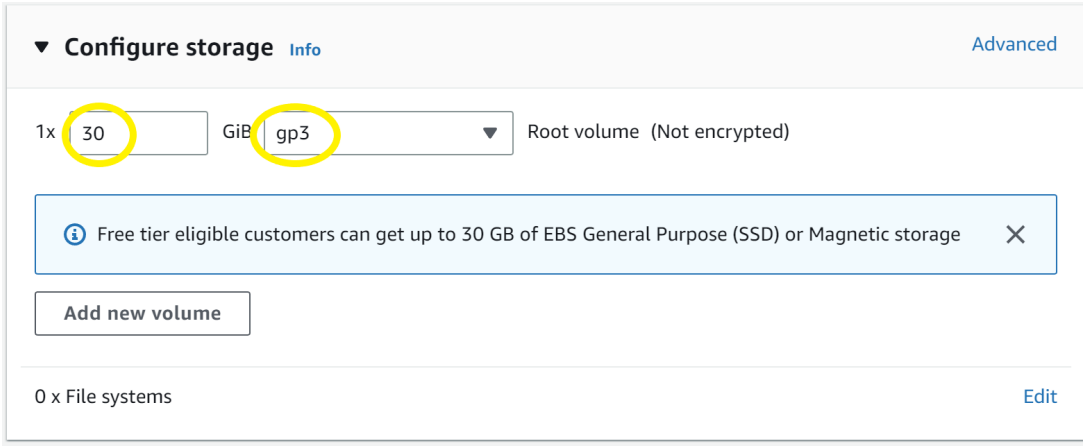
Step 1: Click “**Create Security Group**”
Step 2: Checkmark “***Allow SSH traffic from***”.
Step 3: Dropdown and select “**My IP**”



Step 4: Write down the Security Group name above.
In this case it is “launch-wizard-1”.

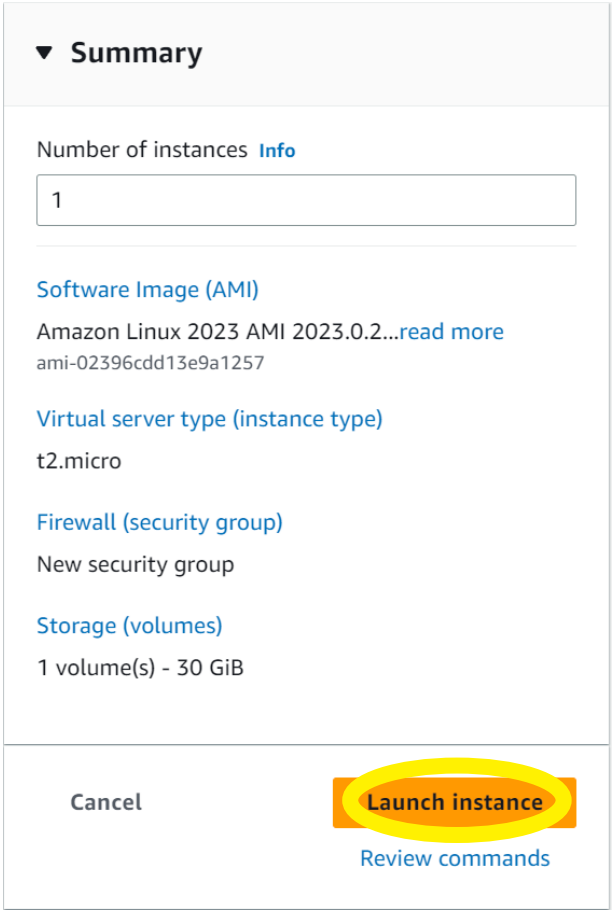
Configure Storage

Step 1: Choose “**30**” GB of hard disk space.
Step 2: Dropdown and choose “**GP3**” Storage Type.



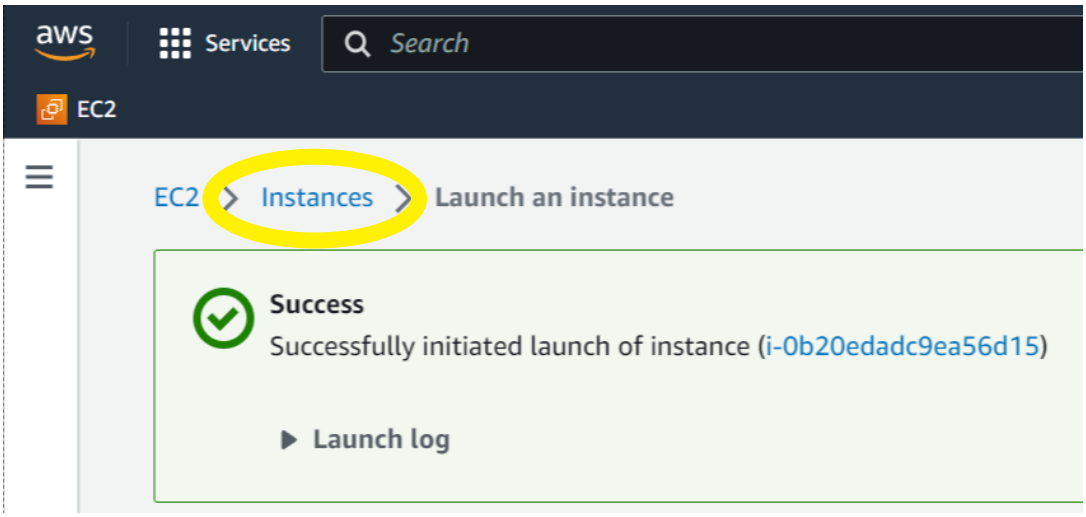
Launch the Instance

Step 1: Verify information is correct
Step 2: Click “**Launch Instance**”



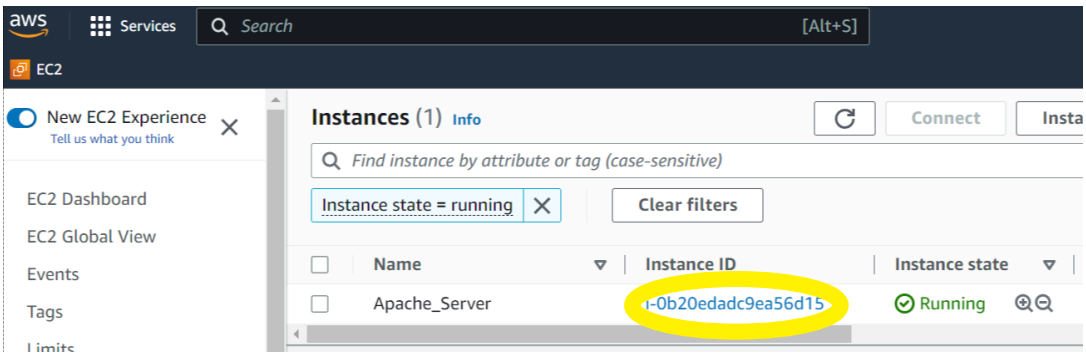
Gathering Server Instance IP Address

Step 1: Click “**Instances**”



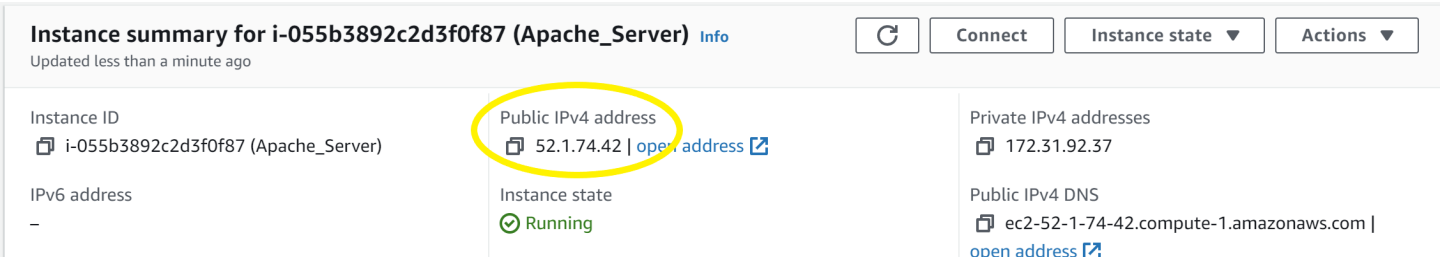
Step 2: Click “**Instance ID**” (Next to the Name you gave the server.)

In the image below it would be the ID “i-0b20edadc9ea56d15”.



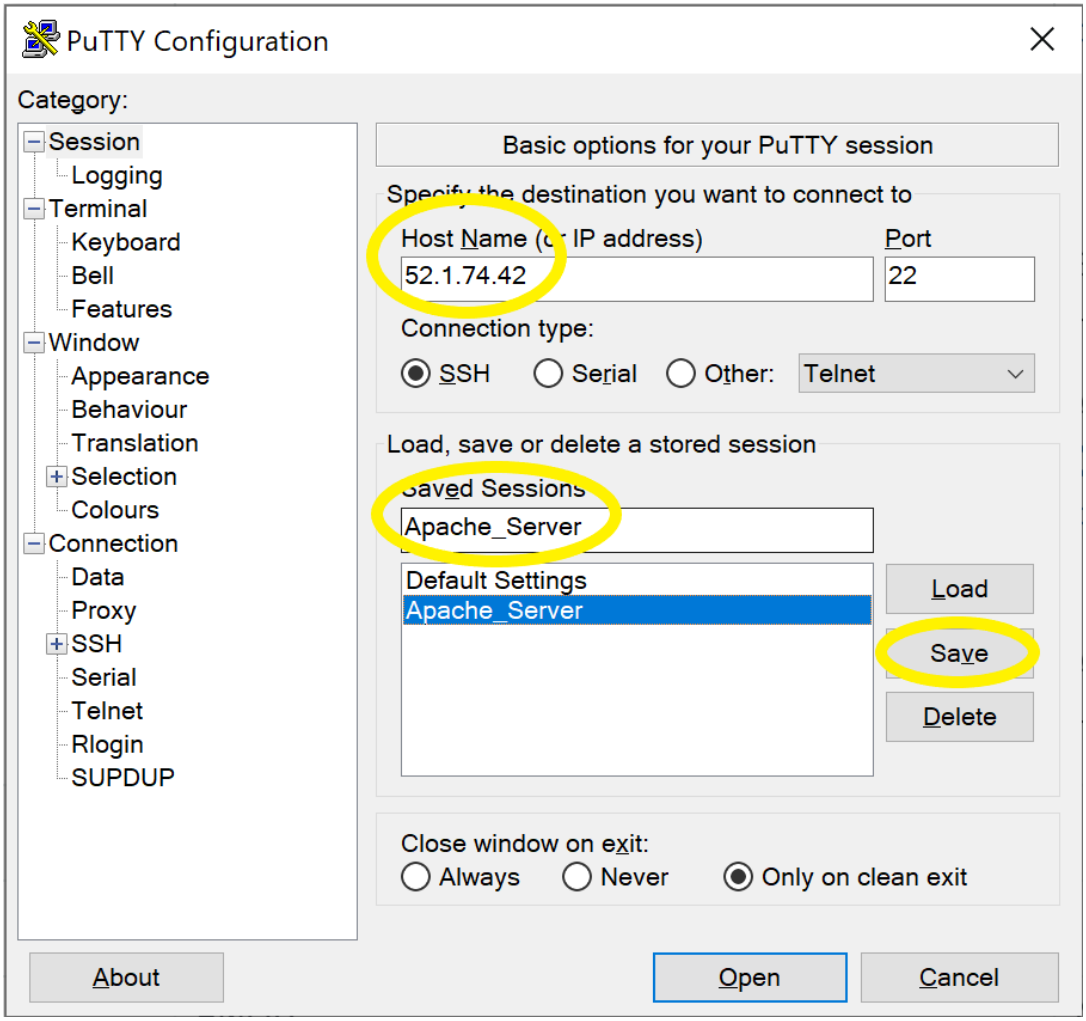
Step 1: Copy the **Public IP** address to your Notepad.

We will use this later.

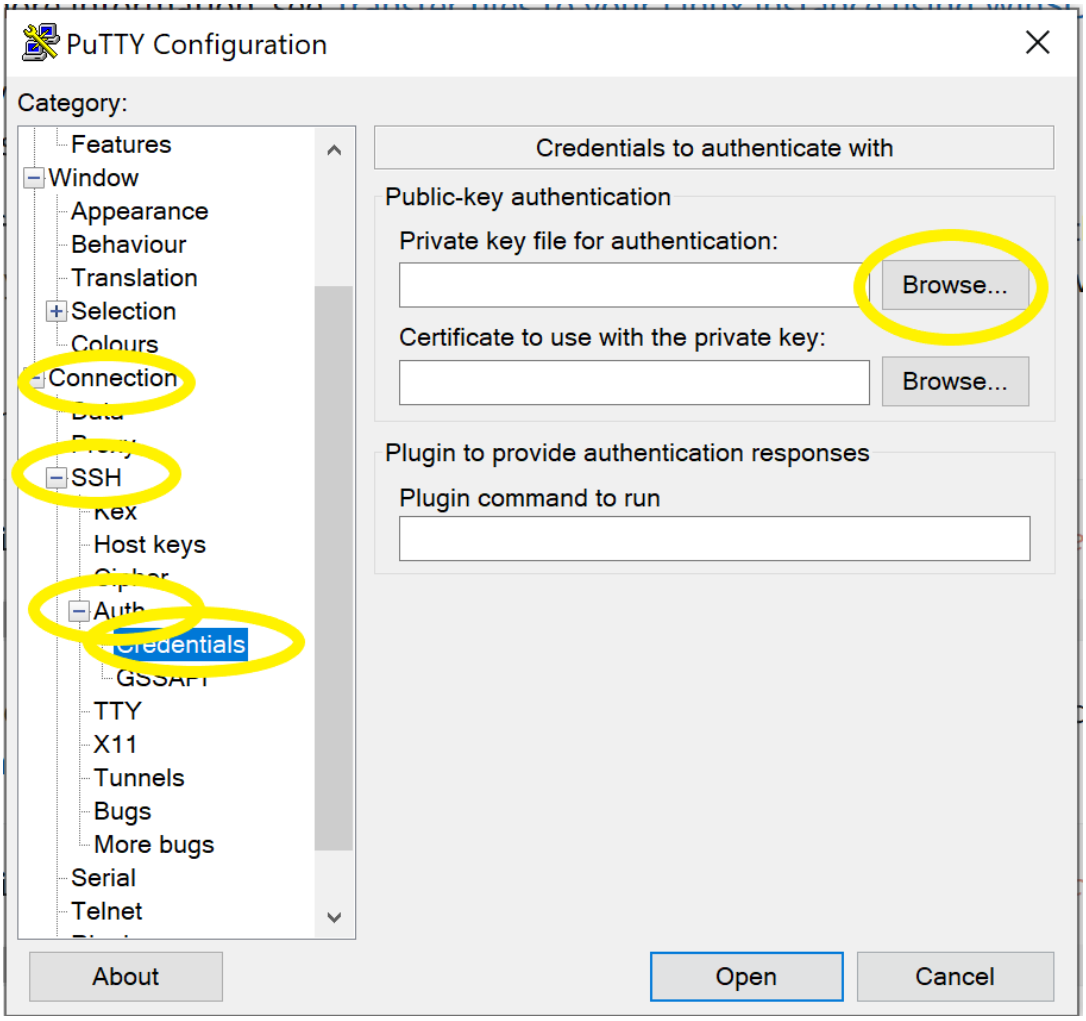


Configure Putty

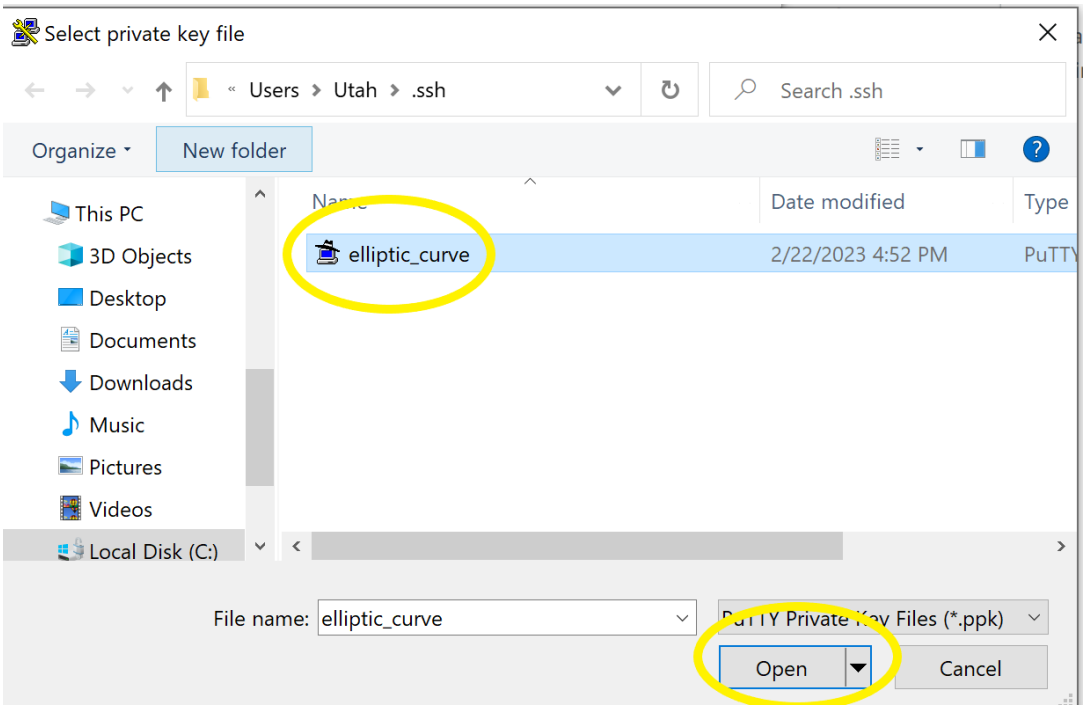
Step 1: Download Putty
https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe
Step 2: Run Putty by double clicking the icon.
Step 3: Copy the **IP Address** we saved into the Host Field of Putty.
Step 4: Enter the **Name** of the server in the Saved Sessions box.
Step 5: Click Save



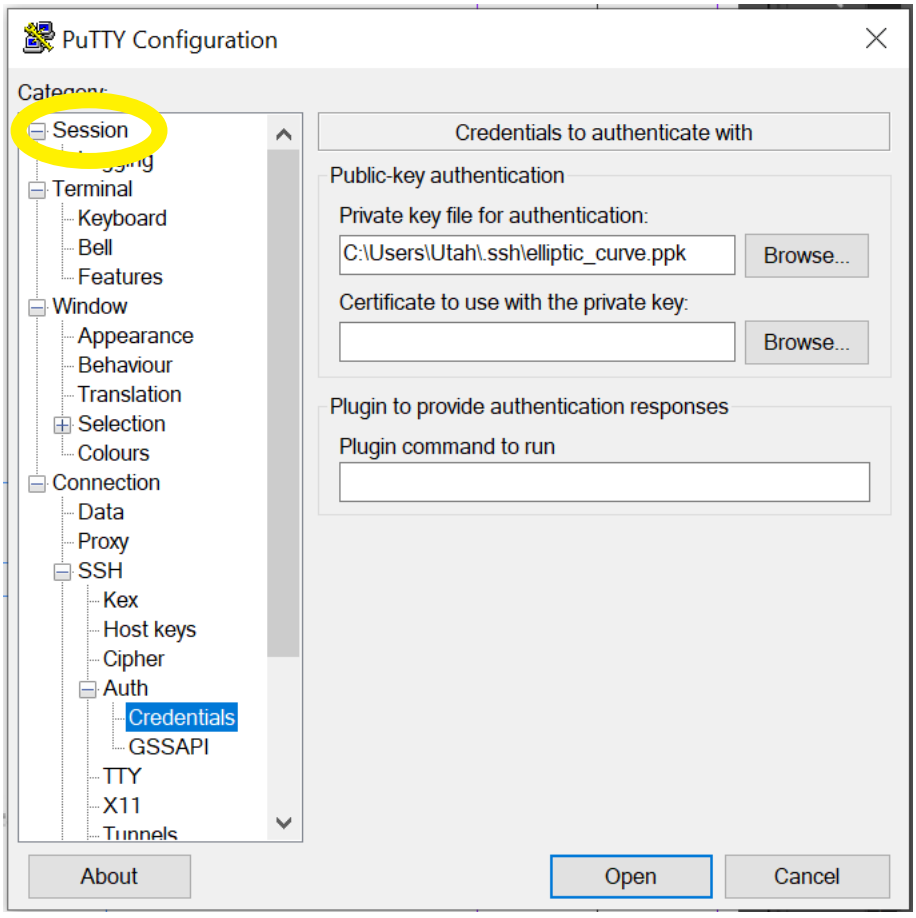
Step 7: Click **Connection** on the left hand side.
Step 8: Expand **SSH**
Step 9: Expand **Auth**
Step 10: Select **Credentials**
Step 11: Click **Browse** under “Private key file for authentication”



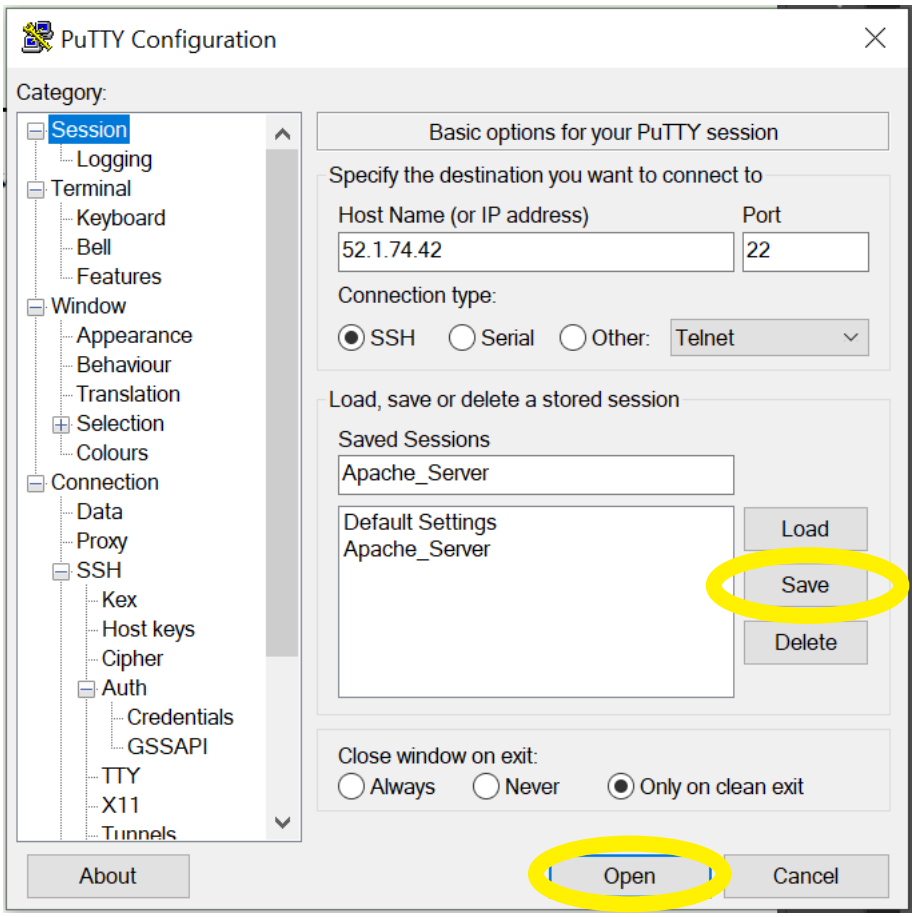
Step 12: Browse to the folder C:\Users***your-username***\.ssh.
Step 13: Choose the Key Pair we created earlier.
Step 14: Select "**Open**".



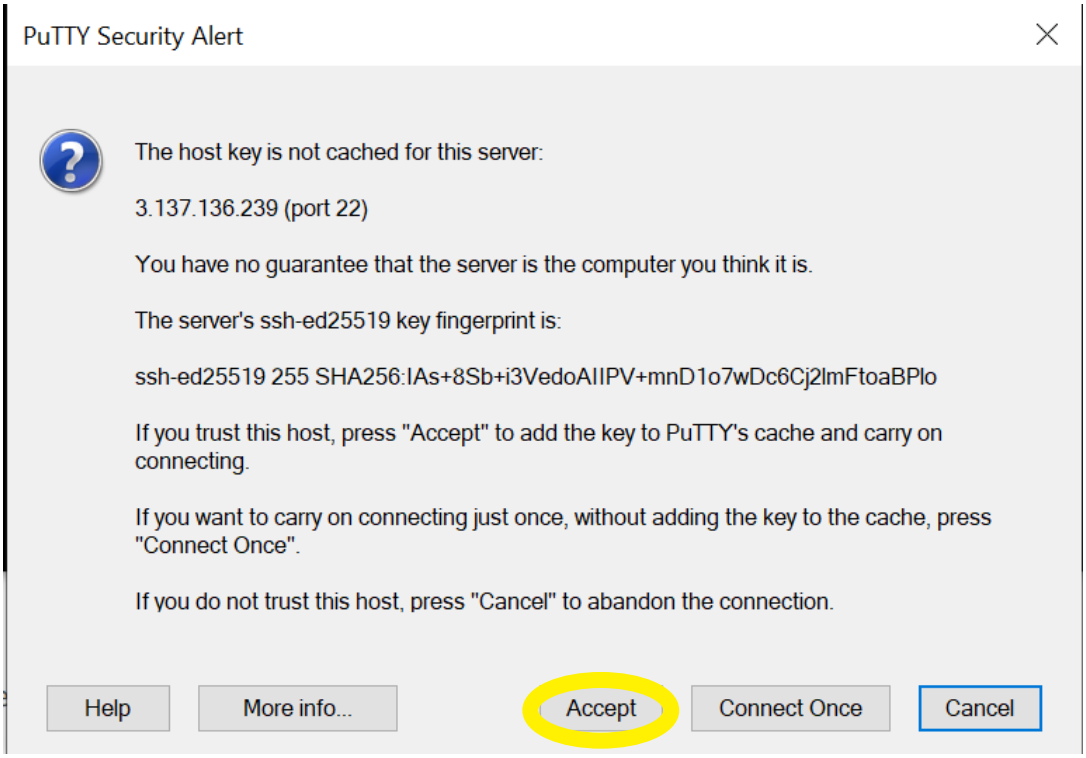
Step 15: Scroll up and Click Session



Step 15: Click “**Save**”
Step 16: Click “**Open**”



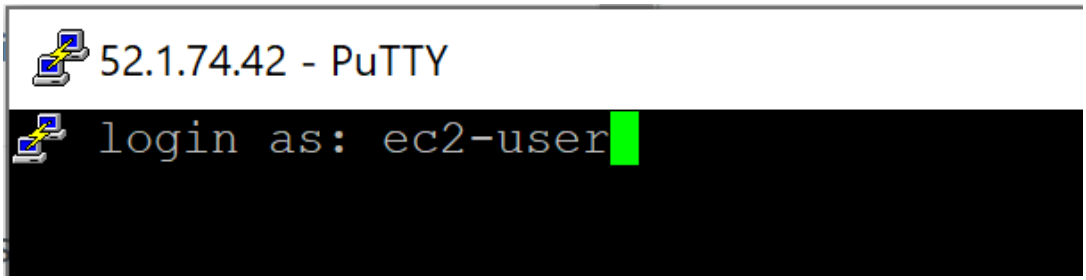
Step 17: Click “Accept”



This may require another re-opening of putty after the accept.

Step 17: Enter Login Name: “**ec2-user**” > Hit **”Enter”**

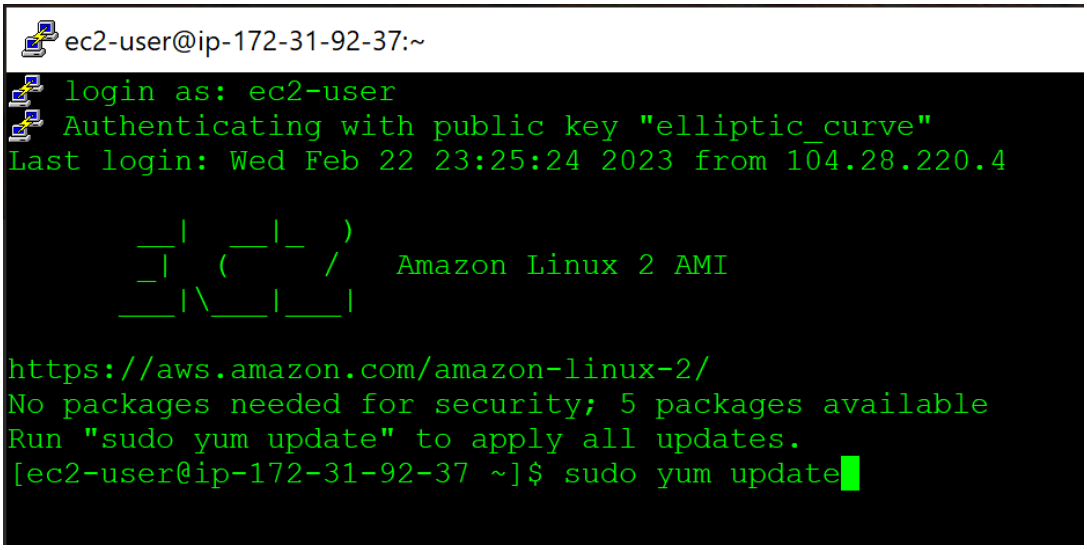
This is the default user for the Amazon Operating System images.
Make sure you it is all lowercase.
Other images such as Ubuntu have different usernames.



Update the OS and Install Apache

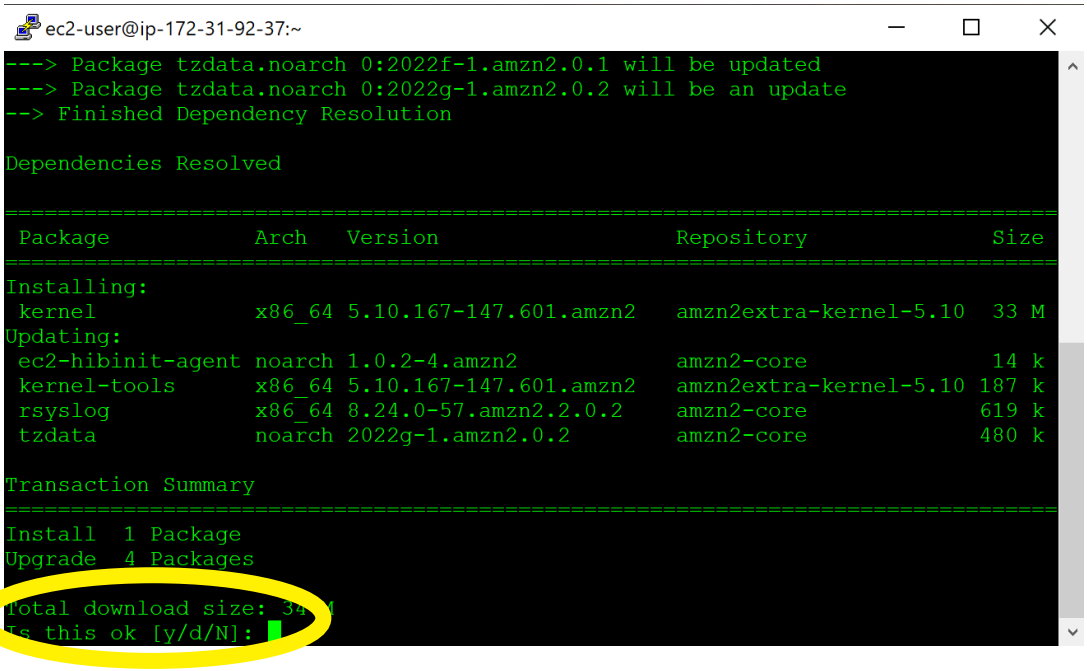
This section shows the process for manually patching and installing a webserver. We call it having servers as pets instead of treating them like cattle. Every server is treated with love and tenderness for patching and maintenance.

Step 1: Type **"sudo yum update"** > Hit **"Enter"**

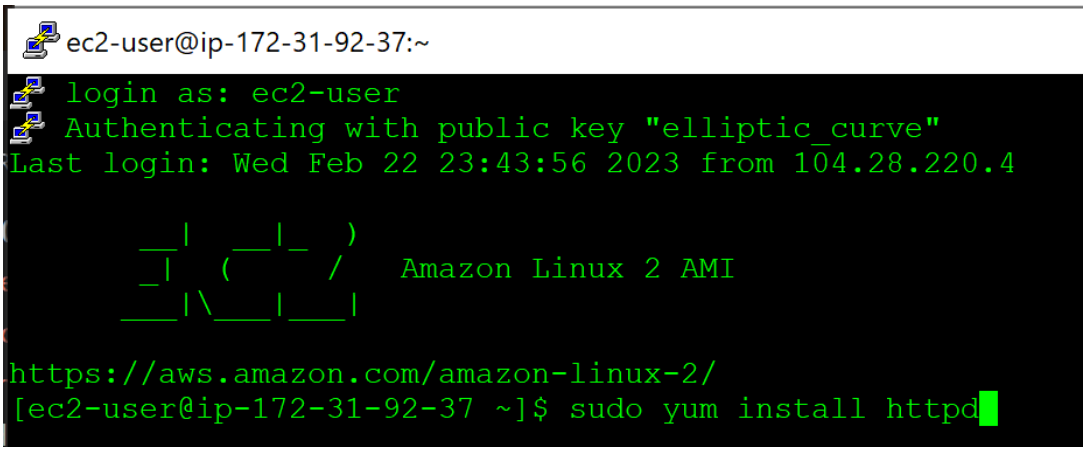


Step 3: Type **"y" for yes**

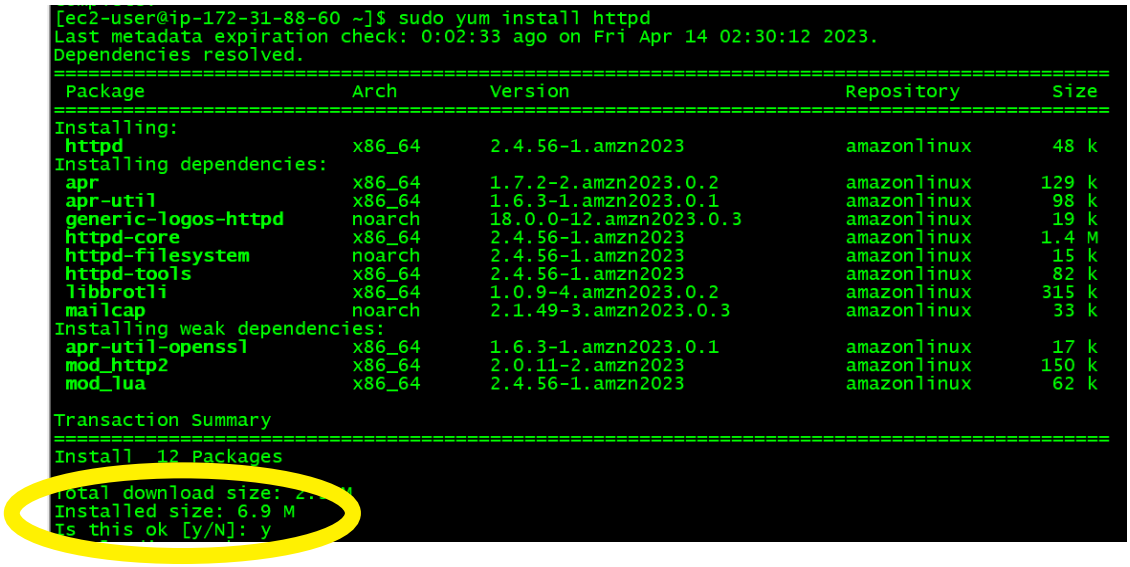
There may be no new patches to install allowing you to skip this step.



Step 4: Type **”sudo yum install httpd”** > Hit **”Enter”**

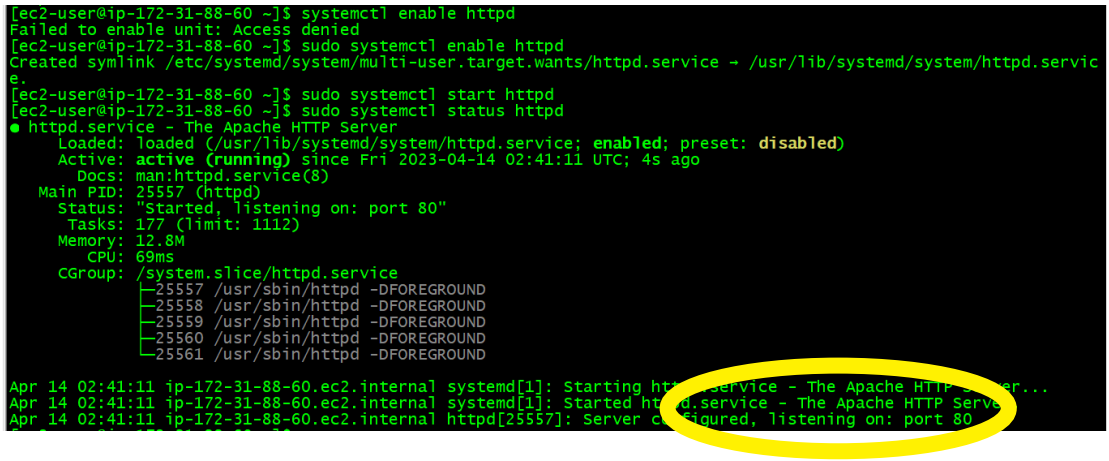


Step 6: Type **”y” to continue installation**



Step 7: Type **”sudo systemctl enable httpd --now”** > Hit **”Enter”**
Step 8: Type **”sudo systemctl status httpd”** > Hit **”Enter”**

Verify that the service is enabled and running with systemctl.

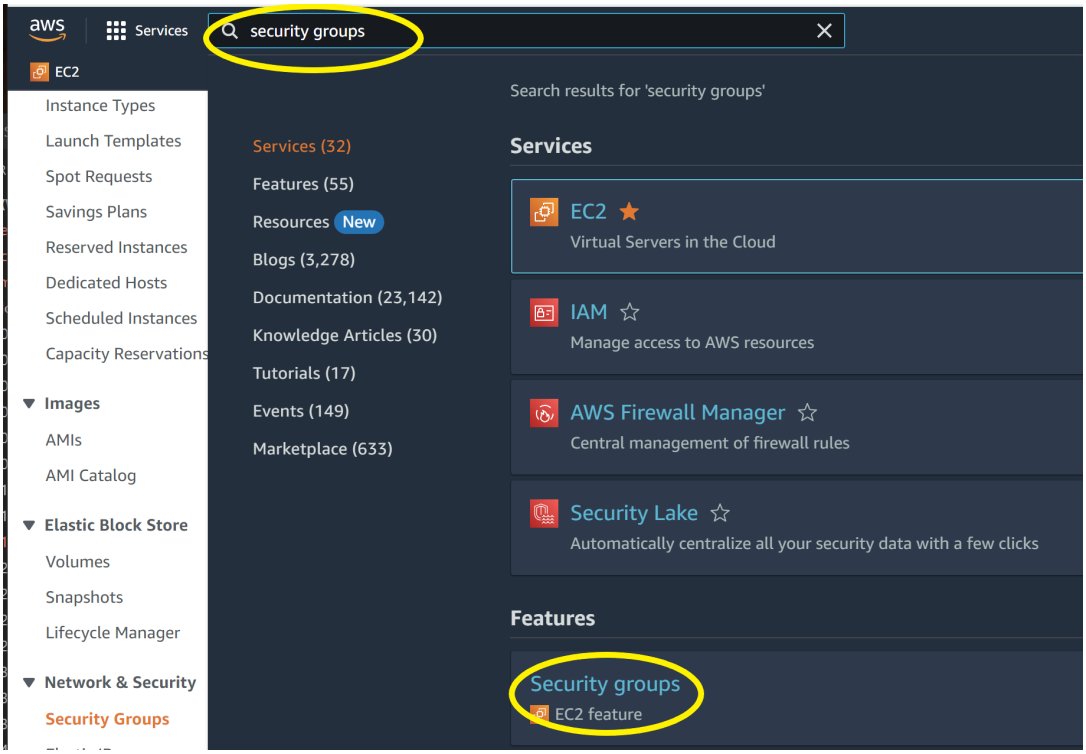


Step 9: Hit **”Control C”** **to exit this screen**

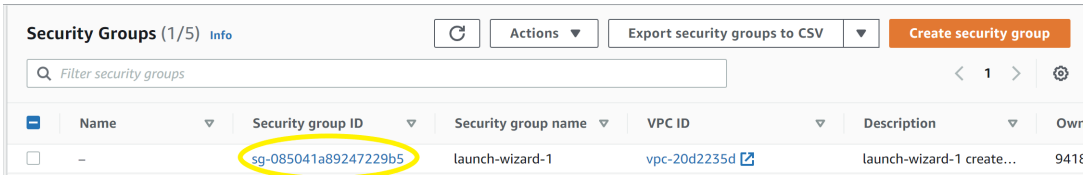
Fix our Security Group to allow HTTP/s

After enabling Apache on the server we need to now go into the security group we made and permit HTTP and HTTPs traffic.

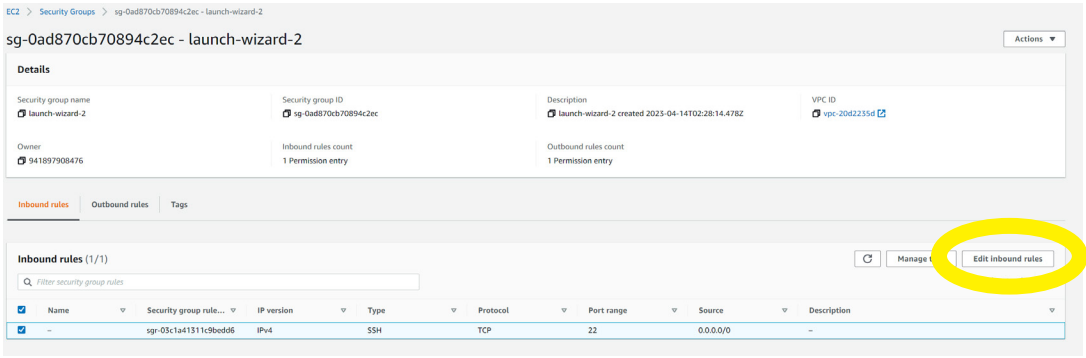
Step 1: Search **”Security groups”**
Step 2: Click **”Security groups”**



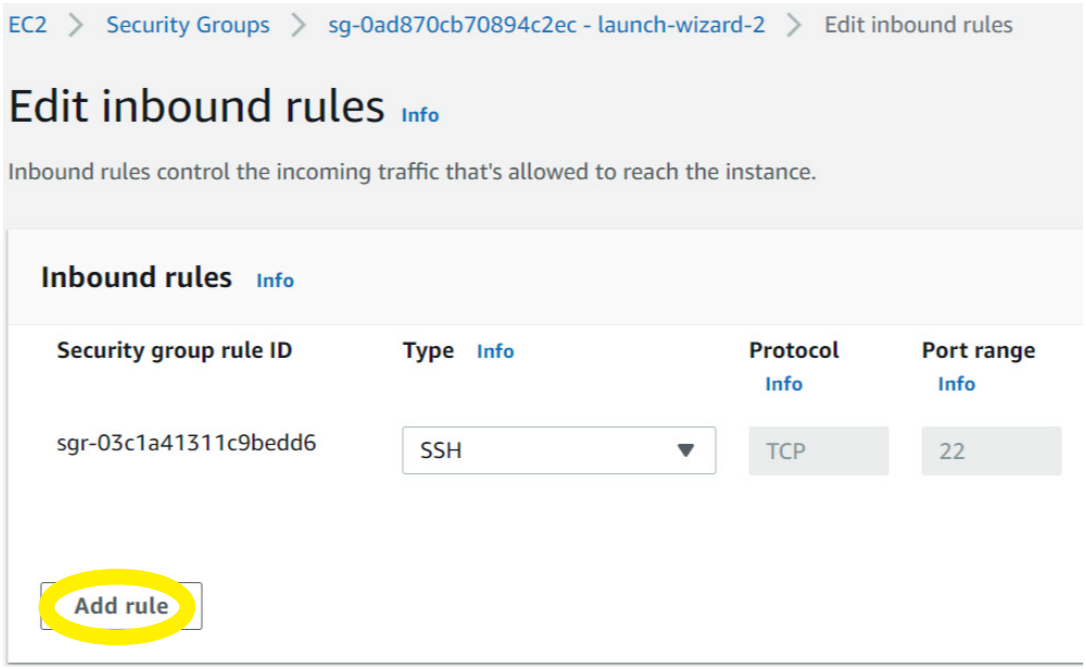
Step 3: Find the **Security Group** we wrote down earlier.
Step 4: Click its **”Security group ID”**



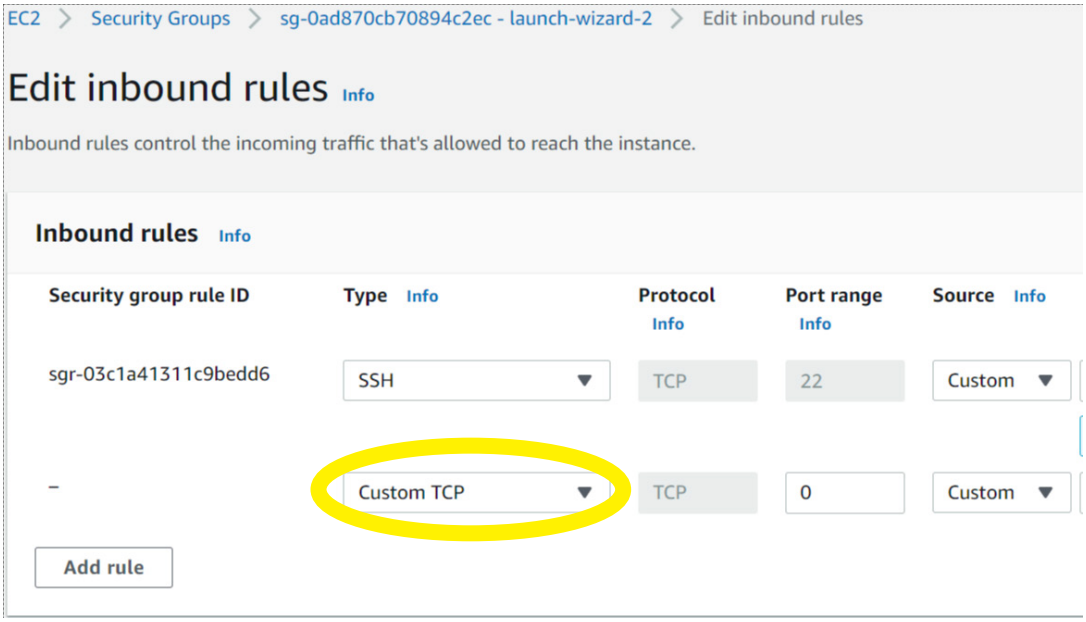
Step 5: Click “Edit Inbound Rules”



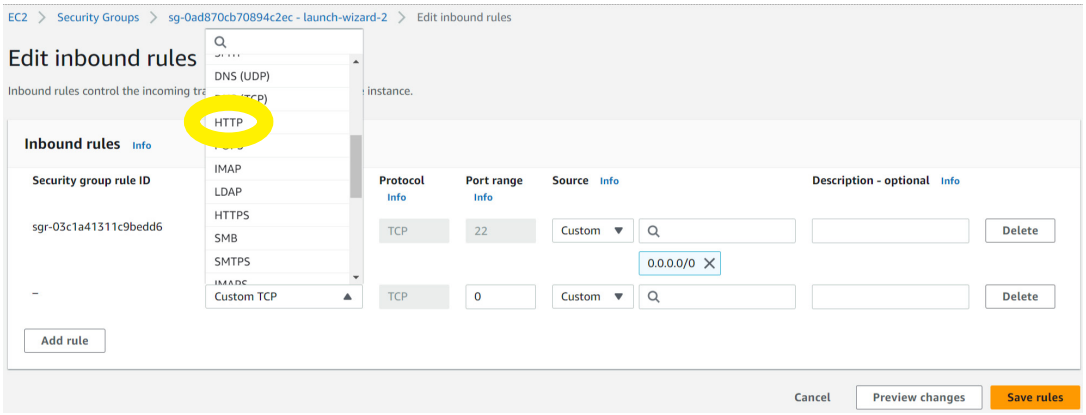
Step 6: Click “Add Rule”



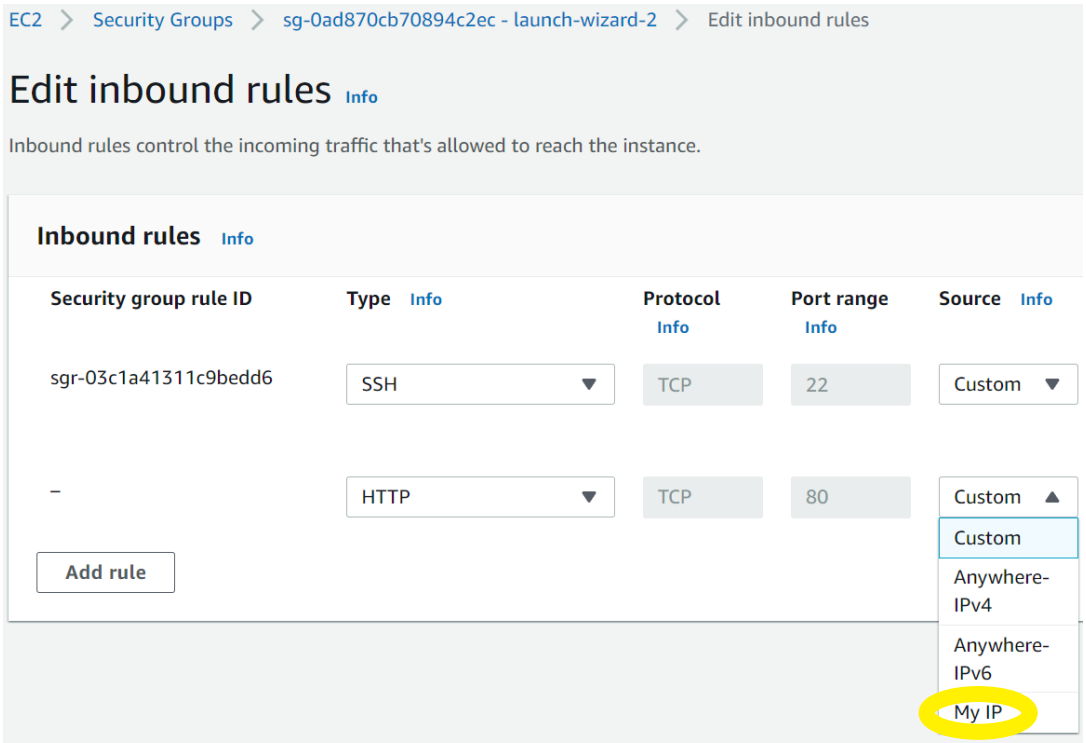
Step 7: Click **“Type Drop Down”**



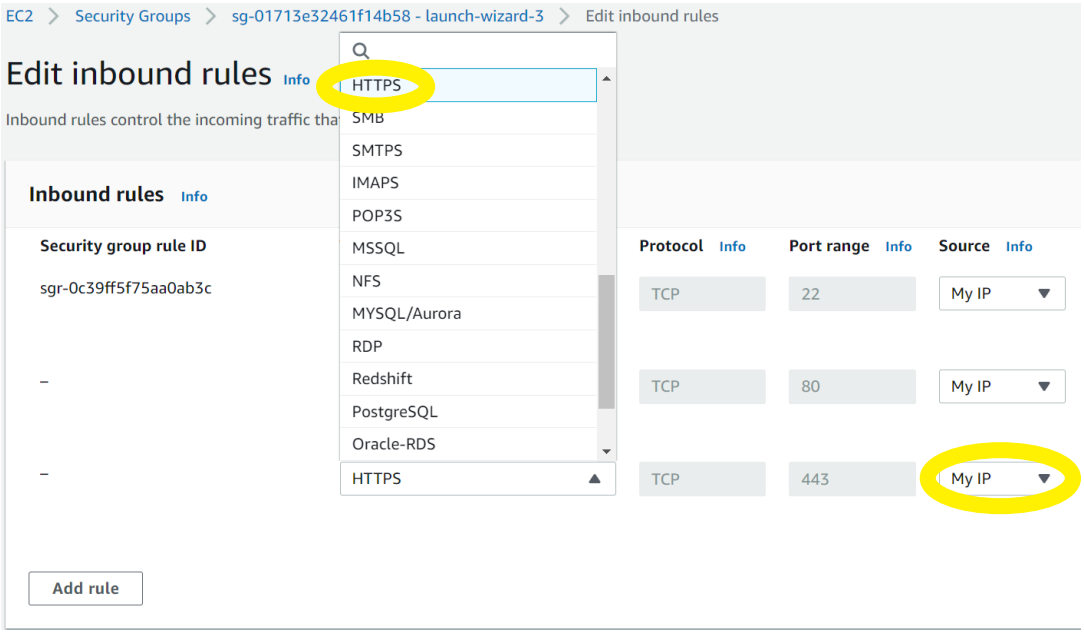
Step 8: Click “HTTP”



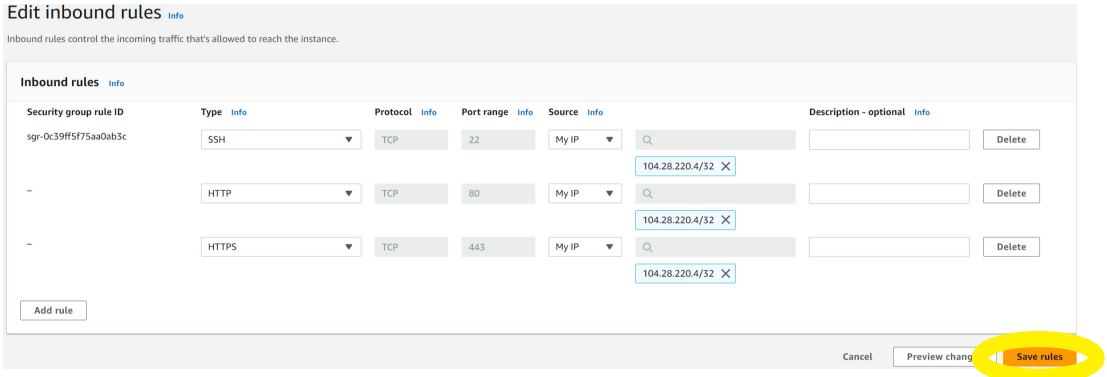
Step 9: Click “**Custom**”
Step 10: Click “**My IP**”



Step 11: Click the “**Type Drop Down**” and select “**HTTPS**”
Step 12: Click “**Custom**”
Step 13: Click “**My IP**”



Step 8: Click “Save Rules”



Install and Modify the Server Software Firewall

We need to install the software firewall on the server as well as open holes for ssh, HTTP, and HTTPs.

Relaunch the Putty Application

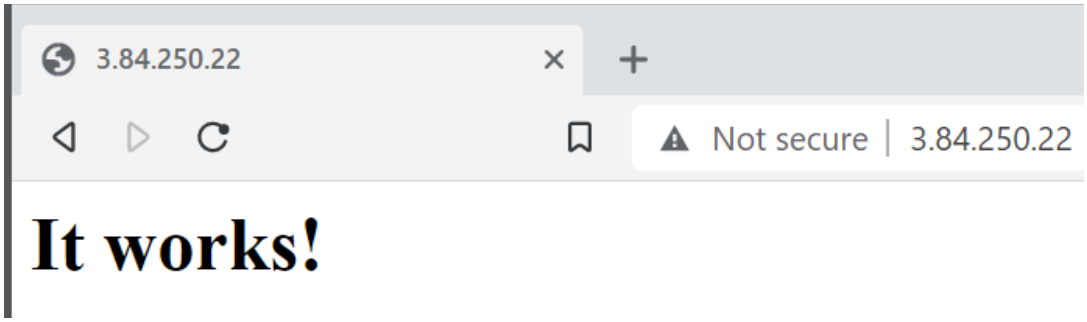
Step 1: Type “sudo dnf --assumeyes install firewalld” > Hit **”Enter”**

```
[ec2-user@ip-172-31-88-60 ~]$ sudo dnf --assumeyes install firewalld
Last metadata expiration check: 0:46:58 ago on Fri Apr 14 02:30:12 2023.
Dependencies resolved.
================================================================================
 Package                  Architecture   Version                  Repository  Size
================================================================================
Installing:
 firewalld                noarch         1.2.3-1.amzn2023         amazonlinux 452 k
Installing dependencies:
 firewalld-filesystem     noarch         1.2.3-1.amzn2023         amazonlinux  11 k
 gobject-introspection    x86_64         1.73.0-2.amzn2023.0.2    amazonlinux 255 k
 ipset                    x86_64         7.11-1.amzn2023.0.3      amazonlinux  40 k
 ipset-libs               x86_64         7.11-1.amzn2023.0.3      amazonlinux  67 k
 iptables-libs            x86_64         1.8.8-3.amzn2023.0.2     amazonlinux 401 k
 iptables-nft             x86_64         1.8.8-3.amzn2023.0.2     amazonlinux 183 k
 libnetfilter_conntrack   x86_64         1.0.8-2.amzn2023.0.2     amazonlinux  58 k
 libnfnetlink             x86_64         1.0.1-19.amzn2023.0.2    amazonlinux  30 k
 libnftnl                 x86_64         1.2.2-2.amzn2023.0.2     amazonlinux  84 k
 nftables                 x86_64         1:1.0.4-3.amzn2023.0.2   amazonlinux 400 k
 python3-firewall         noarch         1.2.3-1.amzn2023         amazonlinux 357 k
 python3-gobject-base     x86_64         3.42.2-2.amzn2023.0.2    amazonlinux 179 k
 python3-gobject-base-noarch noarch      3.42.2-2.amzn2023.0.2    amazonlinux 154 k
 python3-nftables         x86_64         1:1.0.4-3.amzn2023.0.2   amazonlinux  18 k
Installing weak dependencies:
 libcap-ng-python3        x86_64         0.8.2-4.amzn2023.0.2     amazonlinux  30 k

Transaction Summary
================================================================================
Install  16 Packages

Total download size: 2.7 M
Installed size: 11 M
Downloading Packages:
```

Step 2: Type “sudo systemctl enable firewalld --now” > Hit **”Enter”**
Step 3: Type “sudo firewall-cmd --zone=public --permanent --add-service=ssh” > Hit **”Enter”**
Step 4: Type “sudo firewall-cmd --zone=public --permanent --add-service=http” > Hit **”Enter”**
Step 5: Type “sudo firewall-cmd --zone=public --permanent --add-service=https” > Hit **”Enter”**
Step 6: Type “sudo systemctl reload firewalld” > Hit **”Enter”**
Step 7: Type “sudo systemctl status firewalld” > Hit **”Enter”**

```
[ec2-user@ip-172-31-88-60 ~]$ sudo systemctl start firewalld
[ec2-user@ip-172-31-88-60 ~]$ sudo firewall-cmd --zone=public --permanent --add-service=ssh
Warning: ALREADY_ENABLED: ssh
success
[ec2-user@ip-172-31-88-60 ~]$ sudo firewall-cmd --zone=public --permanent --add-service=http
success
[ec2-user@ip-172-31-88-60 ~]$ sudo firewall-cmd --zone=public --permanent --add-service=https
success
[ec2-user@ip-172-31-88-60 ~]$ sudo systemctl enable firewalld
[ec2-user@ip-172-31-88-60 ~]$ sudo systemctl reload firewalld
[ec2-user@ip-172-31-88-60 ~]$
```

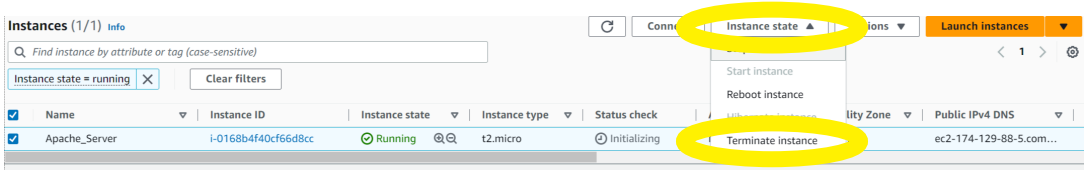
Step 8: Open your web browser and enter the server IP into the address bar.



Delete the Server (Instance)

Deleting all the changes so we do not incur costs. You can also shotdown the instance leaving only storage costs rather than running instance costs.

Step 1: Click "**Instant State**"
Step 2: Click "**Terminate Instance**"



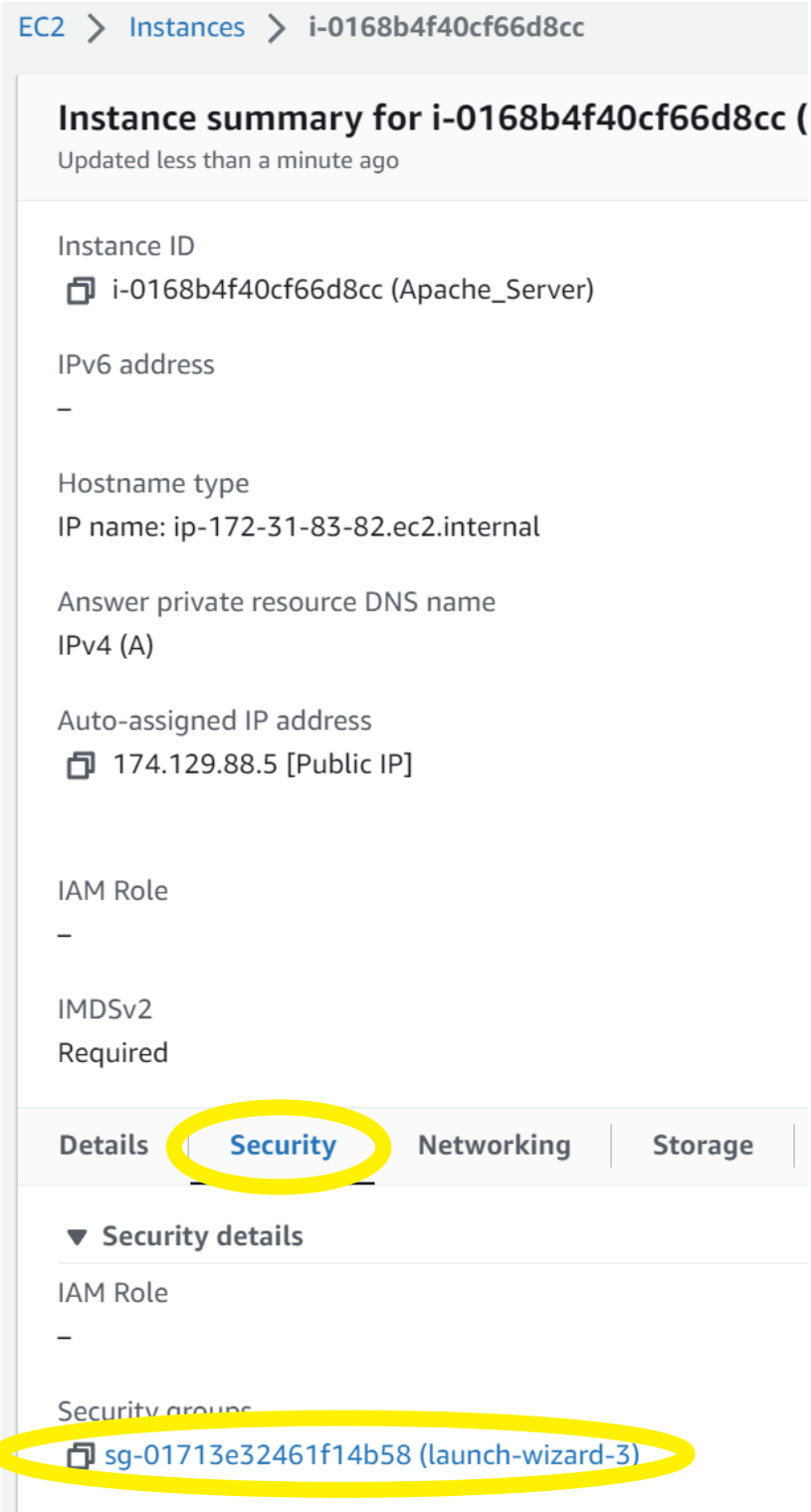
Save the Key-Pair

We will use the Key Pair for Part 2: Infrastructure as Code with Terraform.

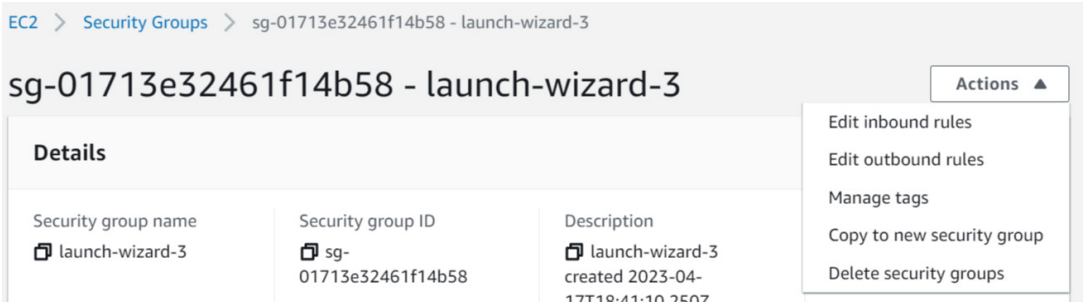
Delete the Security Group

The security group can only be deleted after the Instance has been deleted.

Step 1: Click "**Security**"
Step 2: Click the Security Group in Blue



Step 3: Click “**Action**”
Step 4: Click “**Delete security groups**”



Intro to Cloud 101

Part 2
Infrastructure as Code with Terraform

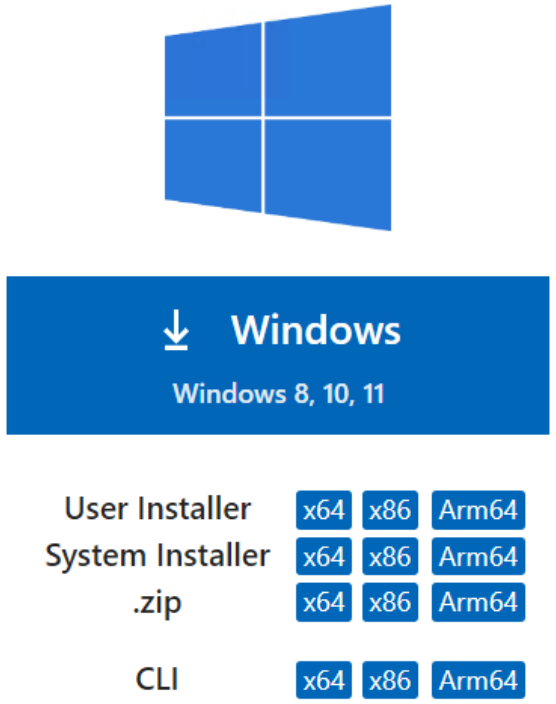
Target Audience: Non Programmers with little to no experience with cloud automation or general cloud use. No previous experience is required.

Disclaimer: This is an ultra simple guide. Breaking the one “terraform.tf” file up into seperate parts is normal practice. It is also normal to use modules. We will not be doing either of those for simplicity sake.

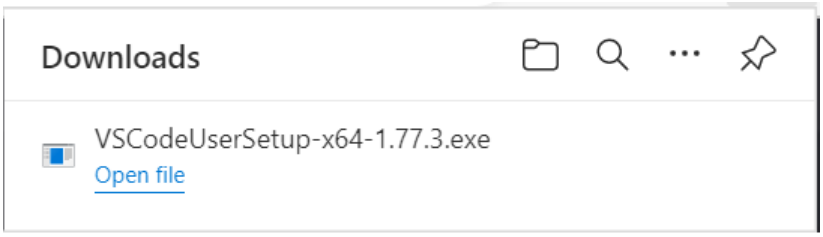
Presenter: David Thurm
Contact:
Linkedin: davidthurm
Twitter: @davidthurm

VScode Install

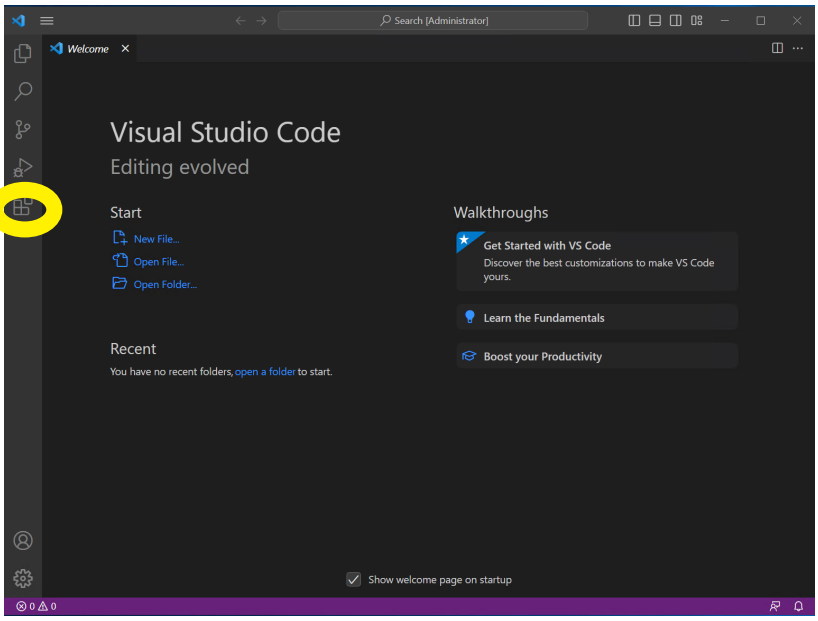
Step 1: Download Code
Step 2: Click “**User Installer x64**”



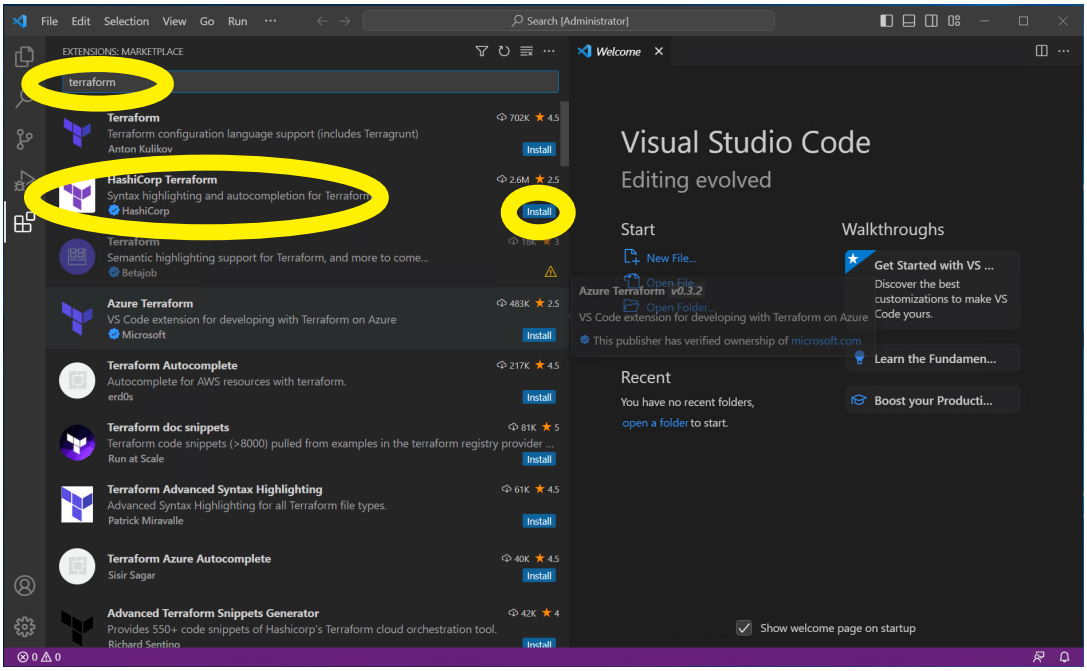
Step 3: Open the Downloaded File



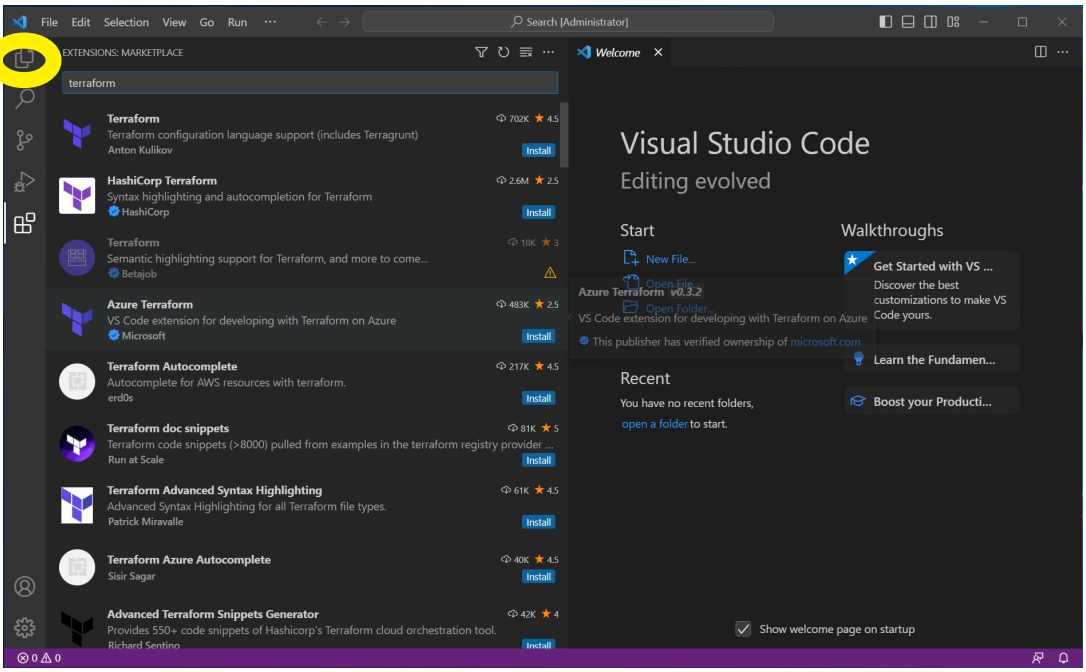
Step 4: Open the Downloaded File
Step 5: Continue as normal as with any software installation.
Step 6: Click “**Extensions**”



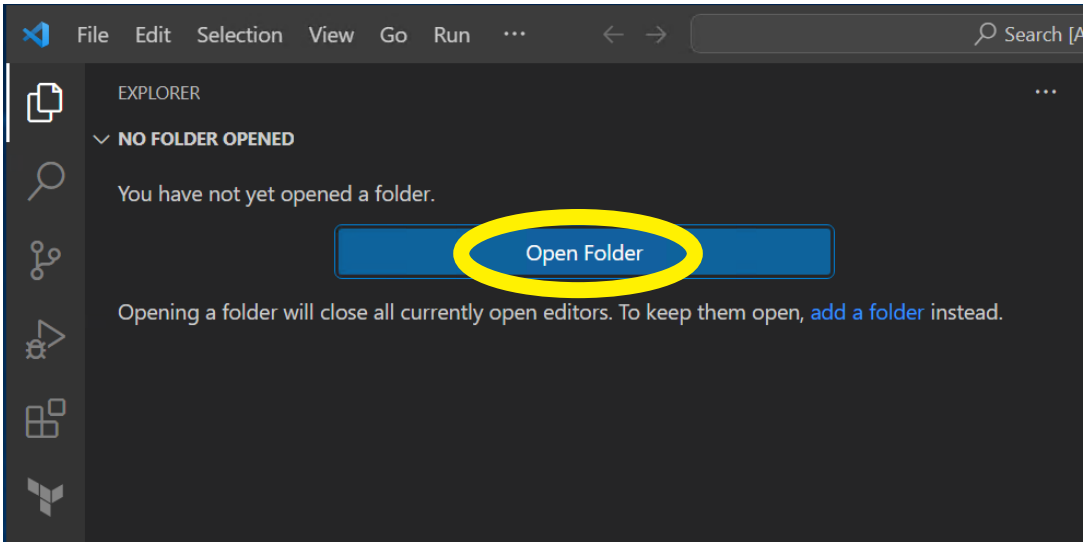
Step 7: Search for “**Terraform**”
Step 8: Verify the vender is HashiCorp
Step 9: Click “**Install**”



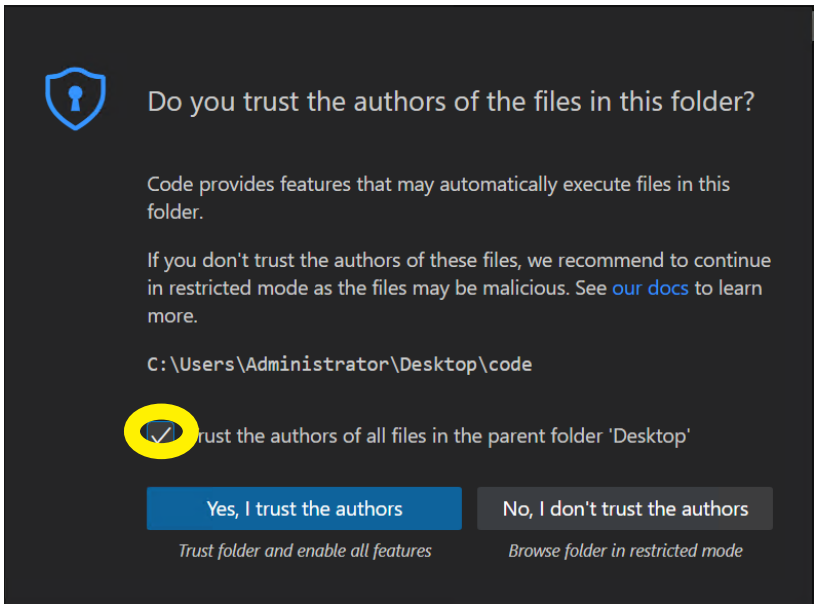
Step 10: Click on “**Explorer**”



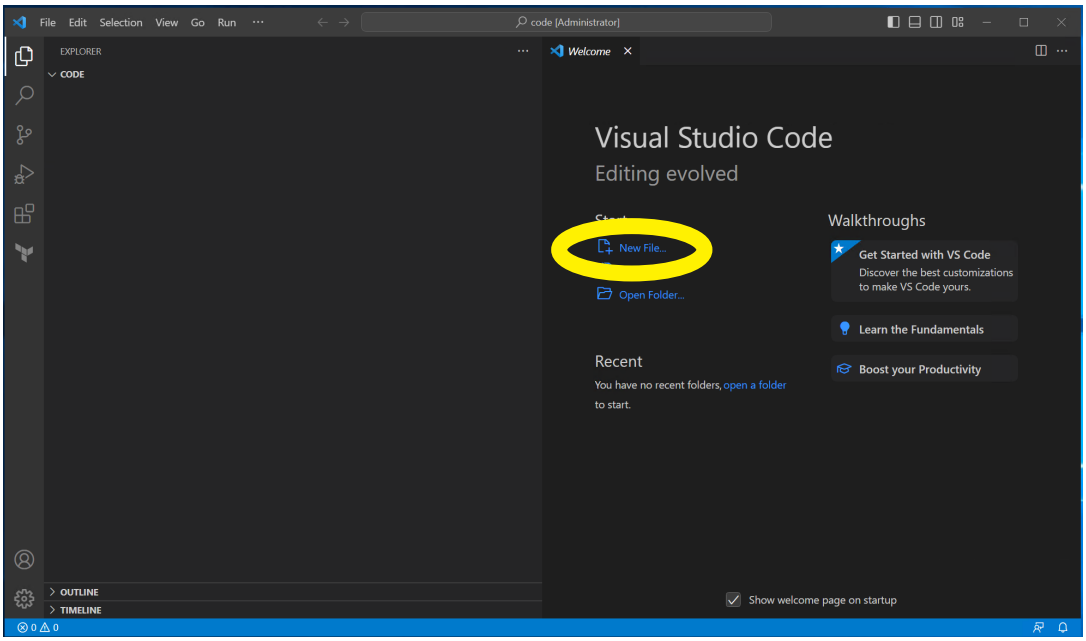
Step 11: Click on “**Open Folder**”
Step 12: Pick a folder of your choice.



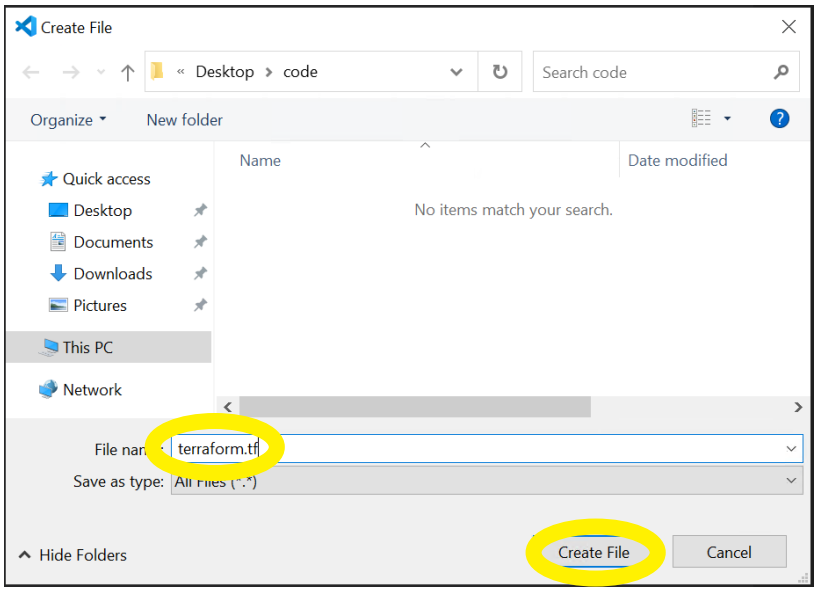
Step 13: Checkmark on “**Trust the authors**”
Step 12: Click “**Yes, Trust the authors**”



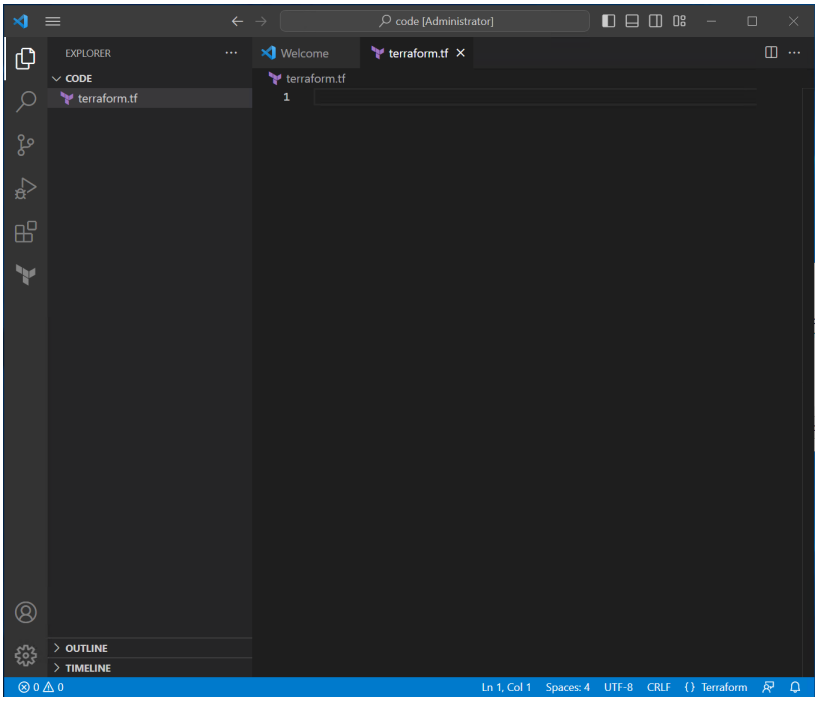
Step 14: Click “New File”



Step 15: Click “New File”
Step 16: Name the File “**terraform.tf**”

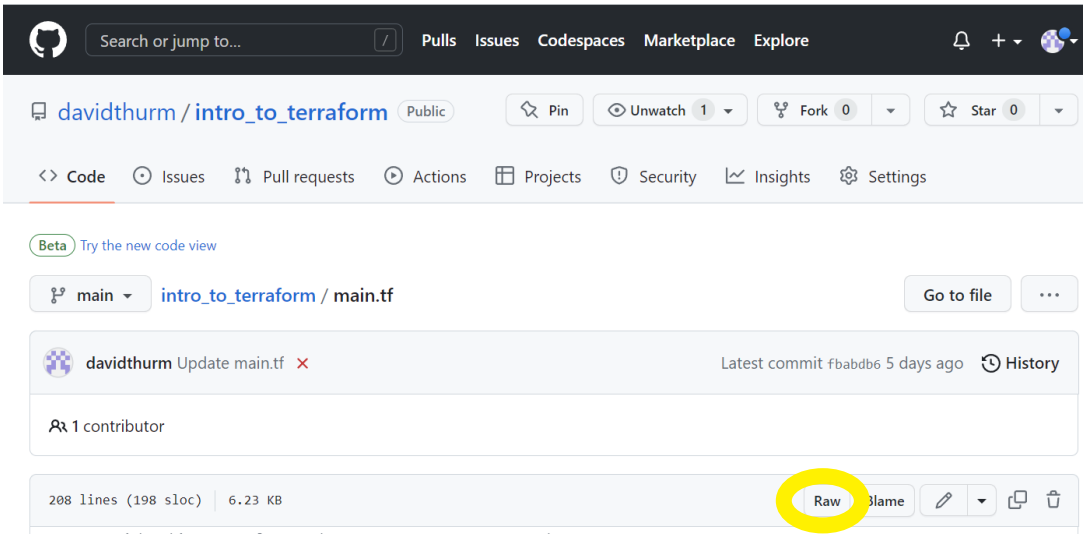


Step 17: We are ready to begin writing Terraform.



Github

Step 1: Browse to https://github.com/davidthurm/intro_to_terraform/blob/main/main.tf
Step 2: Click "**RAW**"



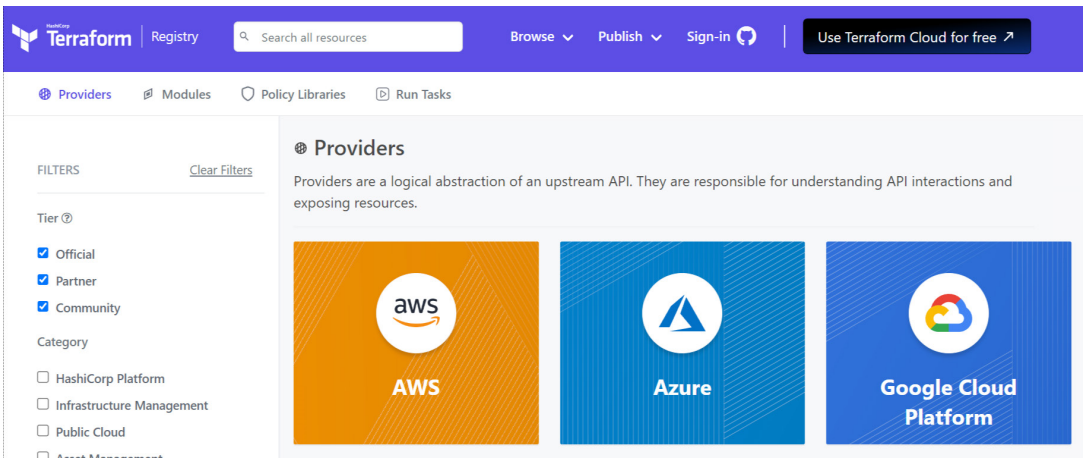
Intentional Bugs are left in this code allowing for changes to be made later on.

Raw is the prefered choice, often times if we do not choose RAW the code will not copy/paste or execute properly when doing CI/CD pipelines. A specific example is using AWS Systems Manager Run Command to launch Ansible code on your servers.

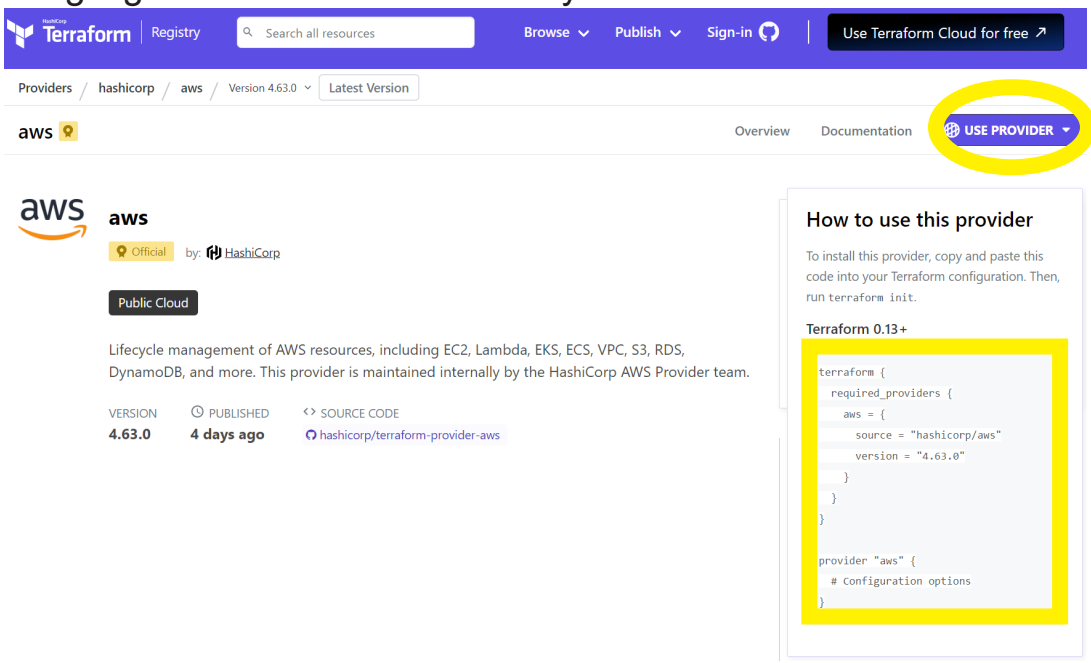
Side-by-Side VScode and Github Repository is recommended for copying code section by section as it is modified during the training.

Pulling down the Terraform AWS Provider Configuration

Step 1: Browse to https://registry.terraform.io/browse/providers
Step 2: Click "**AWS**"



Step 3: Browse to https://registry.terraform.io/browse/providers
Step 4: Click "**USE PROVIDER**"
Step 5: Copy the code highlighted in the Yellow Box to your blank terraform.tf file.



Setting the Manditory Terraform Sections

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "4.62.0"
    }
  }
}

provider "aws" {
  region = "us-east-1" // Change to your desired region
```

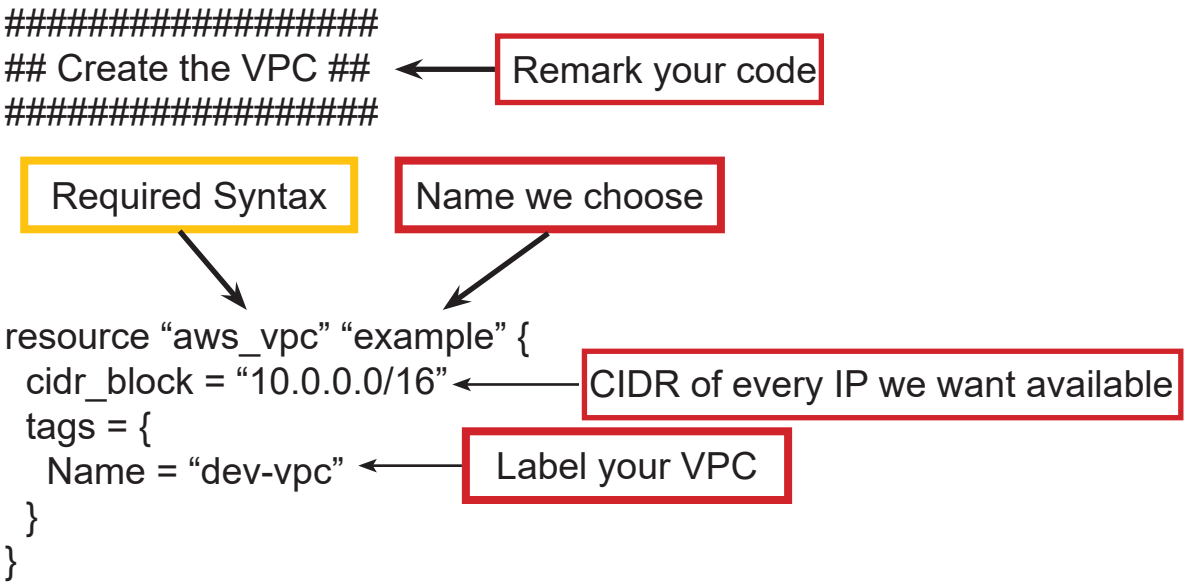
Required

Optional

Terraform Providers are plugins that are required to interact with cloud Platforms.

Provider Regions directly correlate with AWS datacenter regions. Amazon releases newer features into the US East region earlier and is one of the original regions. Not all regions have the same AWS applications and features in them.

VPC Configuration -- Optional (Sometimes)



Remark your code at the top of each section for the next person that comes along.

AWS VPCs are virtual cloud environments. It is comprised of virtual network routers/switches, storage (s3, EBS), web and network firewalls, virtual servers. Imagine your own private workspace.

resource “aws_vpc” identifies that we want to create a VPC.

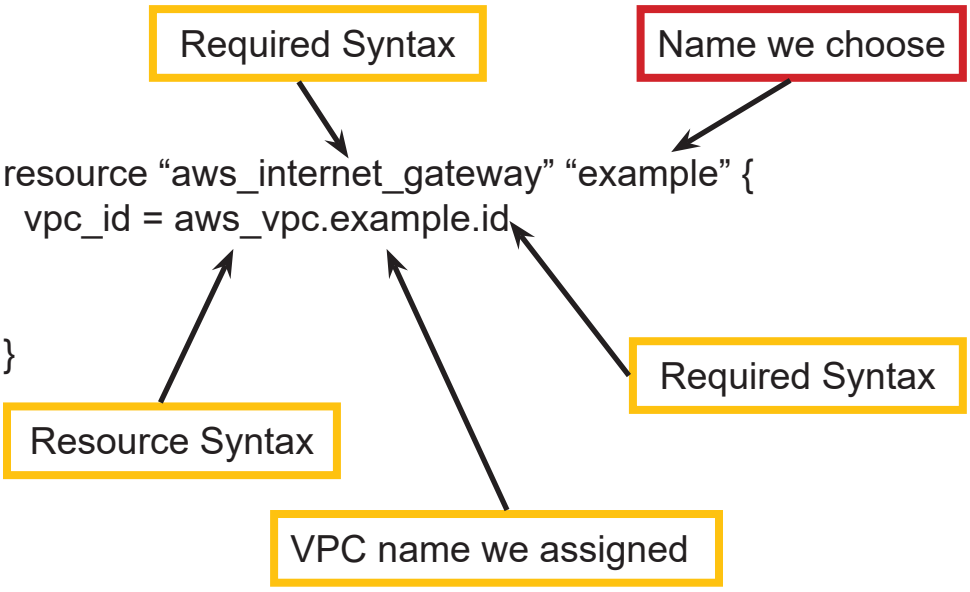
example is a name chosen to describe the VPC. Pick something that is easy to understand and will be repeated throughout the code.

cidr_block is a manditory setting stating available private IP addresses that are going to be inside the VPC.

tags are used to label pieces of the cloud configuration as well as run additional automated tasks against items with a specific tag. They are highly important and it is best to standardize tagging on day one of your cloud migration.

Internet Gateway -- Optional (Sometimes)

```
#############################
## Create the Internet Gateway ##
#############################
```



resource aws_internet_gateway identifies that we will be creating a new gateway the the name of resource name of “example”. Choose your own resource name if you would like. Without this setting our VPC is completely isolated from the Internet and is inaccesable from both inbound and outbound.

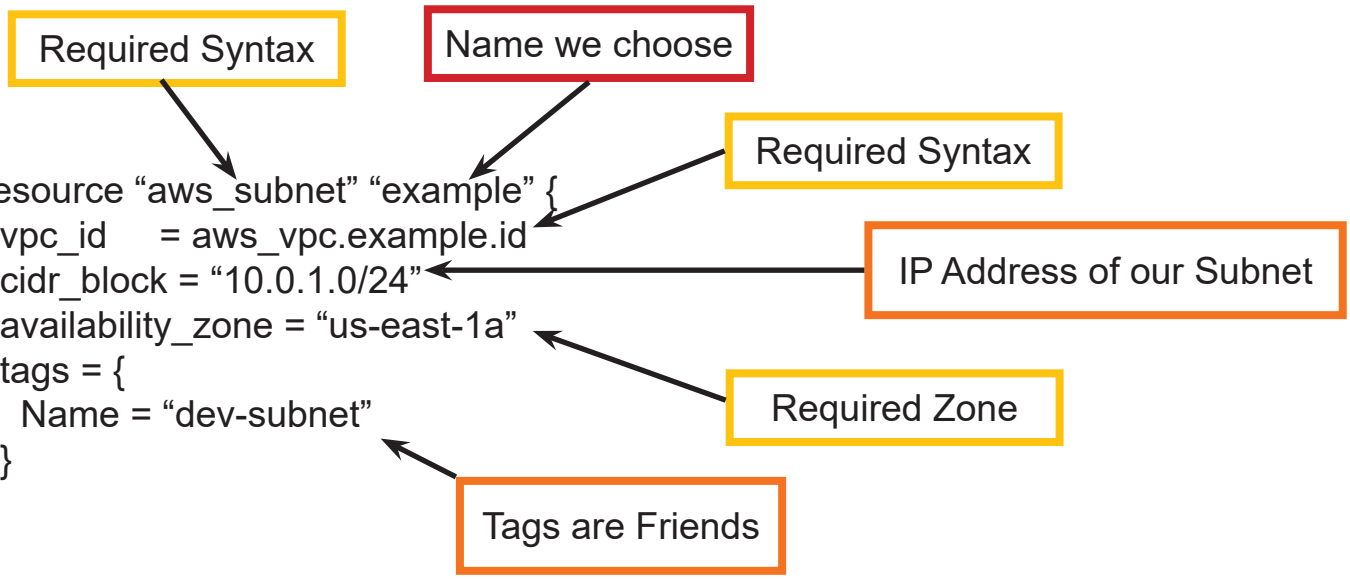
vpc_id refers to the vpc we created above. This is determined by the resource identifier of “aws_vpc” combined with the name we gave the VPC resource and the syntax “id” at the end. The “id” syntax is derived form the vpc_id variable that is automatically generated when we create the VPC.

tags can be added as you see fit.

```
    tags = {
      Name        = “igw-main”
      environment = “dev”
      team        = “platform security”
      owner       = “john@doe.com”
    }
```

Subnet Configuration

```
#####################
## Create the Subnet ##
#####################
resource “aws_subnet” “example” {
  vpc_id     = aws_vpc.example.id
  cidr_block = “10.0.1.0/24”
  availability_zone = “us-east-1a”
  tags = {
    Name = “dev-subnet”
  }
}
```



resource aws_subnet creates the subnet we will use to house our EC2 instance.

vpc_id will match the previous configuration for the Internet Gateway.

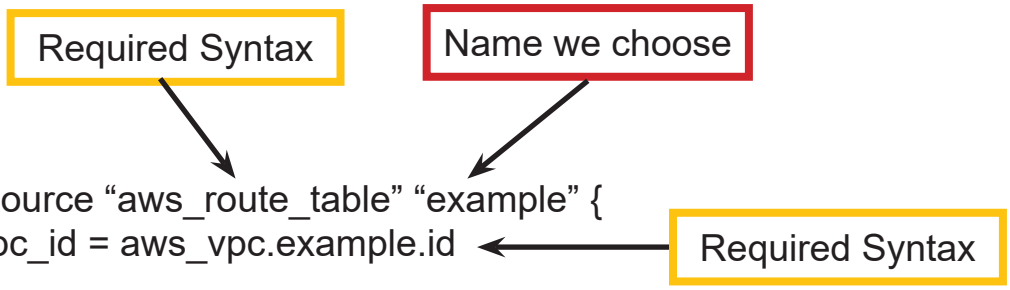
cidr_block this space needs to sit inside the CIDR_block we attached to the VPC we created.

availablity_zone needs to match the zone you wish to deploy your EC2 instance in.

tags as your standards dicate. The setting of “Name” is case sensitive and will show up in the AWS console gui as the name of the resource. This may have changed though.

Global Route Table Configuration

```
#########################
## Create the Route Table ##
#########################
resource “aws_route_table” “example” {
  vpc_id = aws_vpc.example.id

  tags = {
    Name = “example-route-table”
  }
}
```

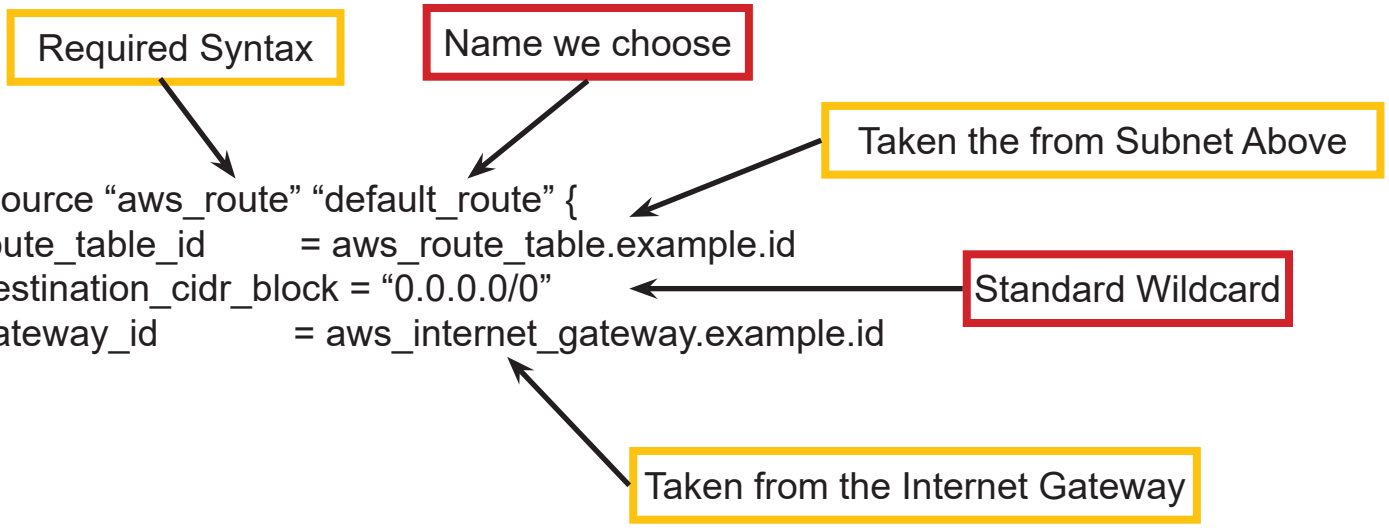


resource aws_route_table creates the default routing table for your VPC. The EC2 instance subnet will be added to this table as well as the default route outbound to the Internet.

vpc_id refers to the vpc we created above.

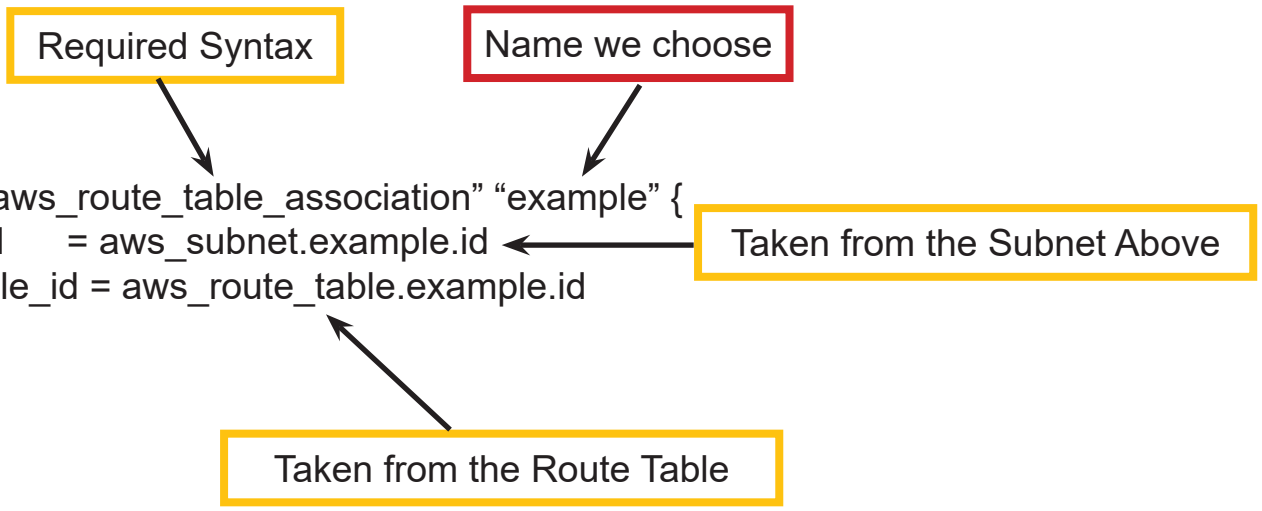
Default Route Configuration

```
############################
## Create the Default Route ##
############################
resource “aws_route” “default_route” {
  route_table_id         = aws_route_table.example.id
  destination_cidr_block = “0.0.0.0/0”
  gateway_id             = aws_internet_gateway.example.id
}
```



Route Association Configuration

```
####################################
## Create the Route Association ##
####################################
resource “aws_route_table_association” “example” {
  subnet_id      = aws_subnet.example.id
  route_table_id = aws_route_table.example.id
}
```



resource aws_route_table_association places the EC2 subnet into the global routing table along with the default Internet Gateway.

Security Group Configuration

```
#########################################
## Create the Security Group to allow SSH  ##
## Modify the cider_blocks to your home IP ##
#########################################
```

Required Syntax

Name we choose

```
resource “aws_security_group” “example” {
  name_prefix = “dev-ssh-access”
```

Name the Security Group

```
  ingress {
    from_port = 22
    to_port   = 22
    protocol  = “tcp”
    cidr_blocks = [“66.0.0.97/32”] # Change this to your home ip
  }
```

Allow Access from IP Space

```
    ingress {
    from_port = 80
    to_port   = 80
    protocol  = “tcp”
    cidr_blocks = [“66.0.0.97/32”] # Change this to your home ip
  }
    ingress {
    from_port = 443
    to_port   = 443
    protocol  = “tcp”
    cidr_blocks = [“66.0.0.97/32”] # Change this to your home ip
  }
```

Standard Destination Port

```
   egress {
    from_port   = 0
    to_port     = 0
    protocol    = “-1”
    cidr_blocks = [“0.0.0.0/0”] ## Allow all outbound traffic
  }
```

Allow Any Port Outbound

```
  tags = {
    Name = “dev-security-group”
  }
```

Standard Wildcard

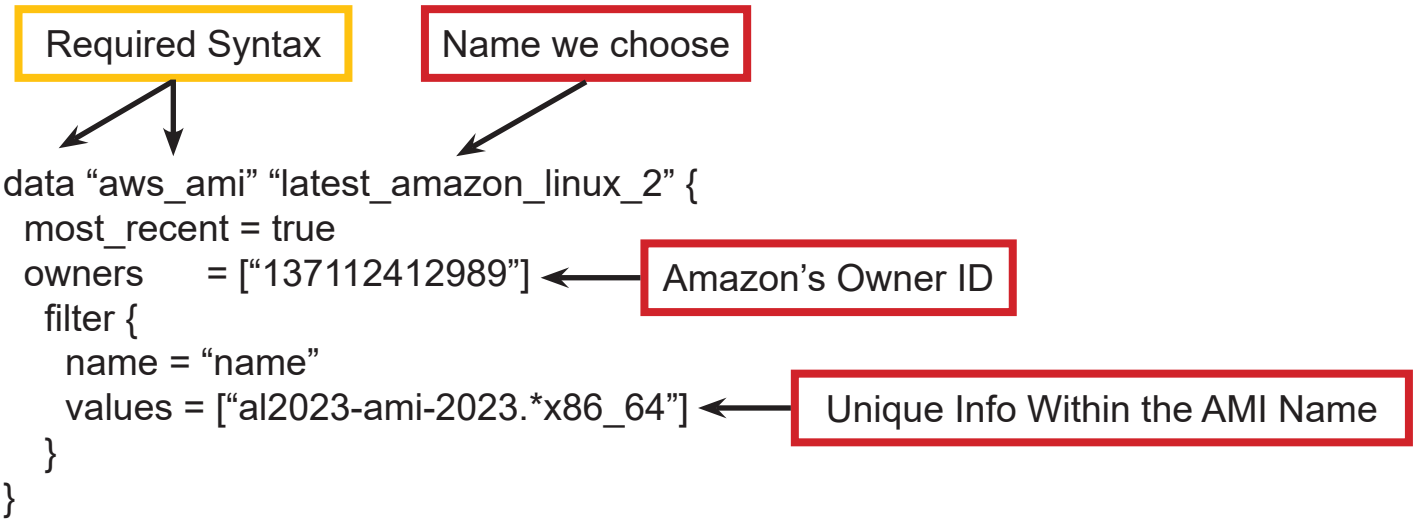
```
  vpc_id = aws_vpc.example.id
}
```

VPC Variable

aws_security_group sets the configuration for the Access Control List (ACL) we want to have in front of our EC2 Instance. This is not a stateful firewall and should be used in addition to a commercial firewall solution in production. It protects traffic inbound and outbound of the Layer 3 subnet. It does not protect traffic between two EC2 instances on the same subnet.

Gather AMI (OS Image) Details

```
####################################
## Search for newest Amazon OS Image ##
####################################
```



data source is used to perform a search of an existing AWS data source. In this case the data source is aws_ami. It will then export the search results under the name “example”.

most_recent will identify what result has the newest version of the image.

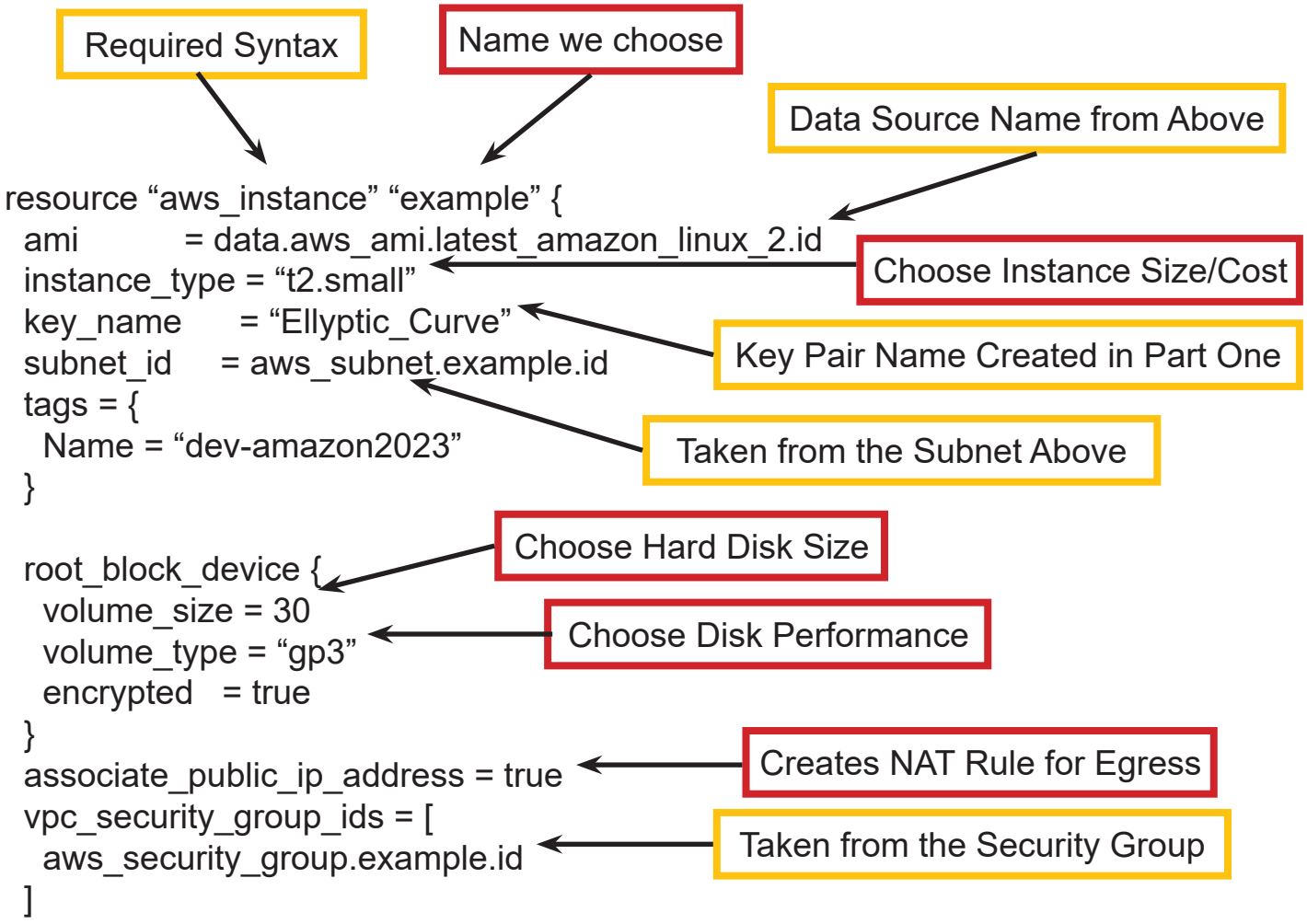
owners identifier is used to spell out that only a specific vendor image is required.

Filter values is created by doing a search via the “aws cli”. It will require several attempts but a starting point is here:

cli command: “aws ec2 describe-images --image-id ami-06e46074ae430fba6”

EC2 Instance Configuration

```
################################
## Create the actual Ec2 Instance ##
################################
```



resource aws_instance initiates the configuration details to be applied to the server. The syntax that follows in this case “example” can be declared as something more useful such as web or fileserver.

ami defines the ami (OS image) we want to deploy. In this case we are using the resource data that we discovered earlier to supply the newest version of Amazon2023 OS ami. This can be substituted for any AMI of your choice by looked up its ID via the AWS web GUI or AWS CLI commands. An example would be a Redhat 9 image “ami-0c41531b8d18cc72b”.

instance_type determines the processing power, network throughput, and amount of memory assigned to your server. This determines **COST**. There are several “Free Tier” instance types available. The first part of the instance_type determines the function of the server. C series servers are for compute intensive workloads while M and T series are general purpose. For most purposes M and T series are preferred giving a balance of compute, memory, and networking. Running servers for limited periods of time does not incur a significant costs as long as they are shutdown when not in use. Feel free to research spot intances for further cost reduction. Information on pricing can be found here: https://aws.amazon.com/ec2/pricing/on-demand/

root_block_device defines the size and disk type needed for the EC2 Instance. In the example we are deploying 30 GB of disk space on General Purpose SSD Volumes. While servers are turned off Storage Costs are still charged.

encrypted only takes affect when the server is turned off and at rest. This does not protect from data theft while the instance is running.

User Data Configuration

```
  user_data = <<EOF
#!/bin/bash
### Standard Patching
sudo dnf --assumeyes update

### Install Firewalld ###
sudo dnf --assumeyes install firewalld
sudo systemctl enable firewalld --now
sudo firewall-cmd --zone=public --permanent --add-service=ssh
sudo firewall-cmd --zone=public --permanent --add-service=http
sudo firewall-cmd --zone=public --permanent --add-service=https
sudo systemctl reload firewalld

### Install Nginx ###
sudo dnf --assumeyes install nginx
sudo systemctl enable nginx --now

### Configure Nginx ###
sudo rm /usr/share/nginx/html/index.html
sudo touch /usr/share/nginx/html/index.html
sudo chown -R  nginx:nginx /usr/share/nginx/html

sudo cat > /usr/share/nginx/html/index.html << EOF1
<html>
   <head>
      <title>Welcome to Test Website!</title>
   </head>
   <body>
      <p>It works!  Thank you for visiting</b>!</p>
   </body>
</html>
EOF
}
```

user_data is used for configuring and installing applications during server initial bootup. In this example we update the Linux repositories, patch the server, install a software firewall, configure the firewall to allow in web traffic and ssh, install nginx, and create a static webpage. Further configurations can be accomplished via AWS Systems Manager.

Output AMI Info

```
#####################################
## Print the Public IP to the Console Screen ##
#####################################

output “public_ip” {
  value = aws_instance.example.public_ip
}
#################################
## Print the AMI info out into a variable ##
#################################

output “latest_amazon_linux_2_ami_id” {
  value = data.aws_ami.latest_amazon_linux_2.id
}
```

Display Server’s Public IP after Apply

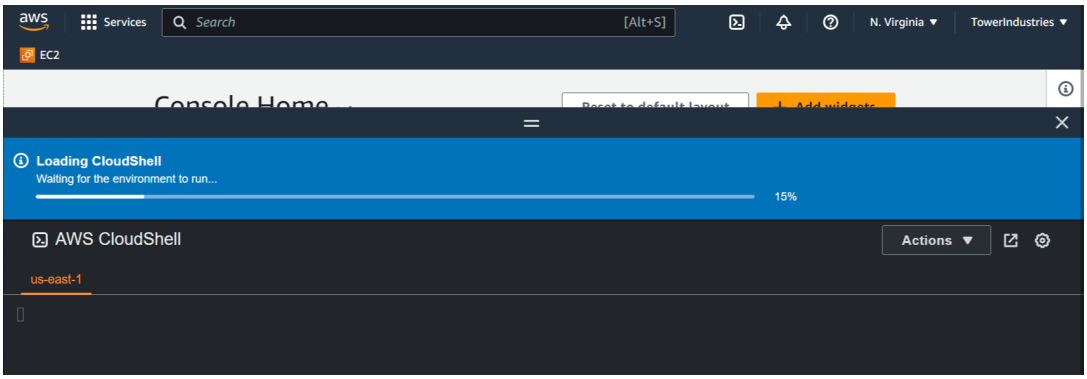
Display Server’s AMI Image After Apply

output displays on screen any setting we would like upon applying the terraform plan. In this instance it will display the public IP of the server saving time when configuring Putty for remote access. As part of personal preference the AMI that was chosen using our data resource is also displayed.

Installing Terraform in CloudShell

Step 1: Click “**CloudShell Icon**”

Step 2: Wait for CloudShell to boot.



Authentication should be transparent when deploying terraform from the cloud shell. In the event this does not work configure here: IAM > Users > “Your Username” > Security Credentials > Access Keys > Create Access Key > Command Line Interface > Click “I Understand” box > Next > Key Description > Create Access Key

Access Key ID is a publicly available identifier that uniquely identifies the AWS account associated with the access key.

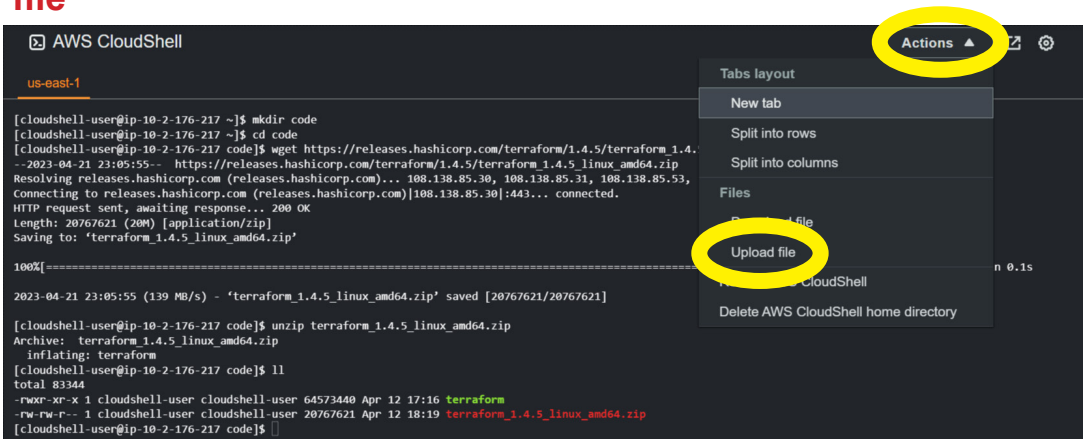
Secret Access Key is a private key used to sign requests made to AWS services.

AWS configure command will initate the login process from the cloud shell cli.

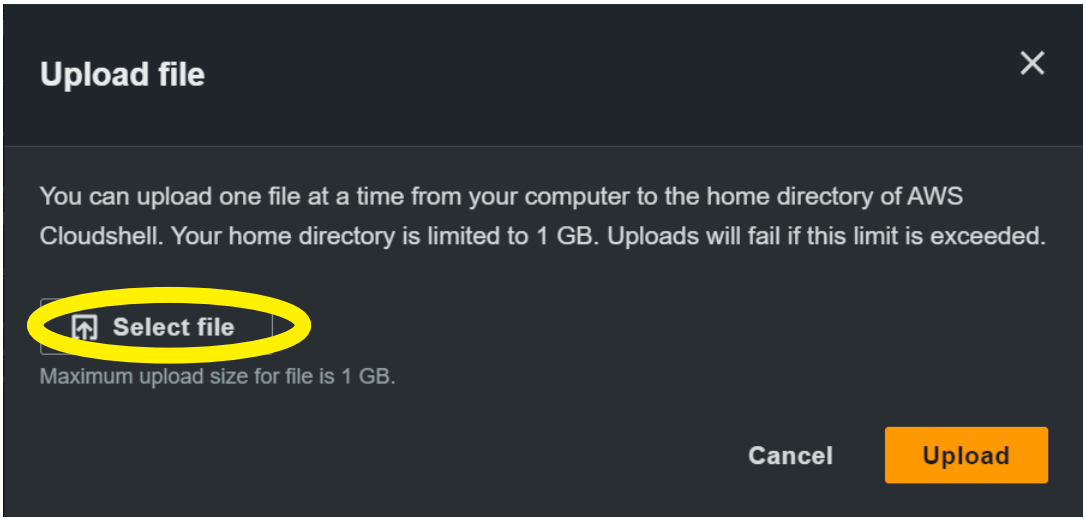
Step 3: Type **”mkdir code”** > Hit **”Enter”**
Step 4: Type **”cd code”** > Hit **”Enter”**
Step 5: Type **”wget https://releases.hashicorp.com/terraform/1.4.5/terraform_1.4.5_linux_amd64.zip”** > Hit **”Enter”**
Step 6: Type **”unzip terraform_1.4.5_linux_amd64.zip”** > Hit **”Enter”**
Step 7: Type **”rm terraform_1.4.5_linux_amd64.zip”** > Hit **”Enter”**

Upload Terraform.tf

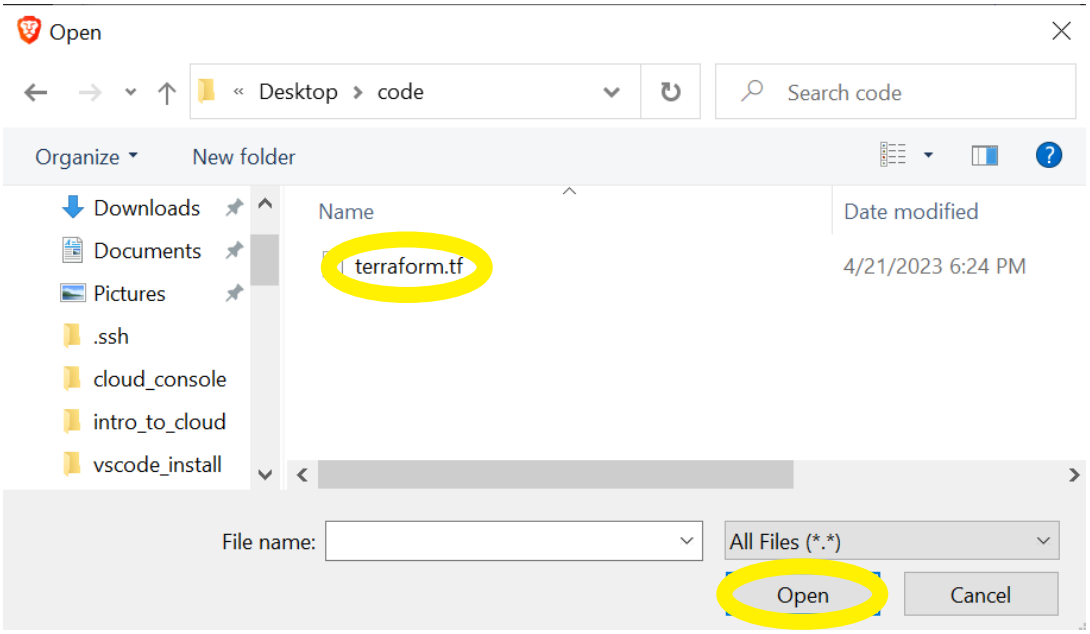
Step 1: Click “**Click Actions**”
Step 2: Click “**Upload file**”



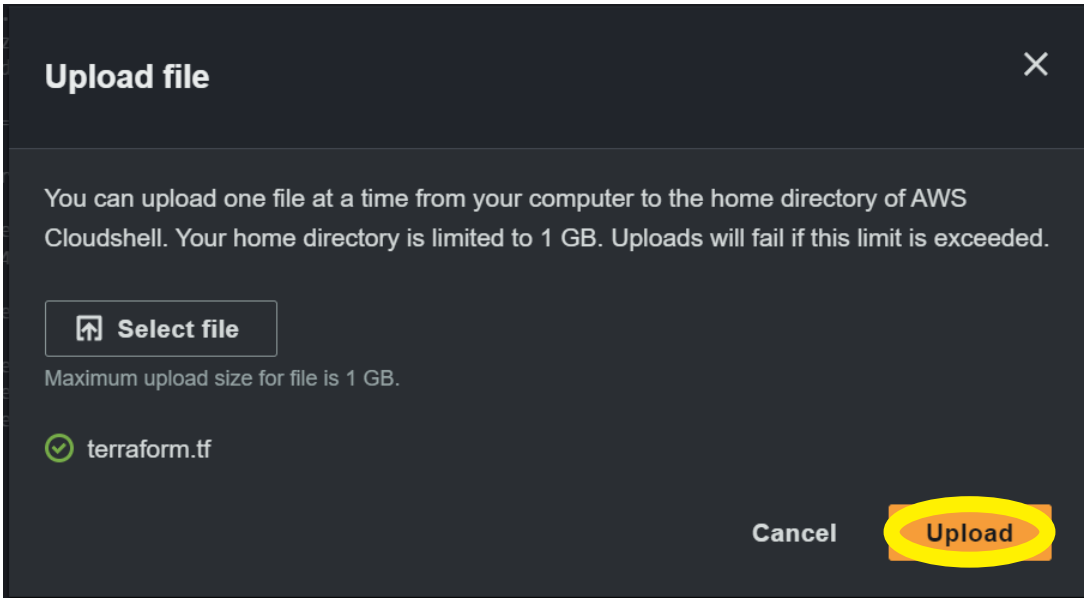
Step 3: Click “**Select File**”



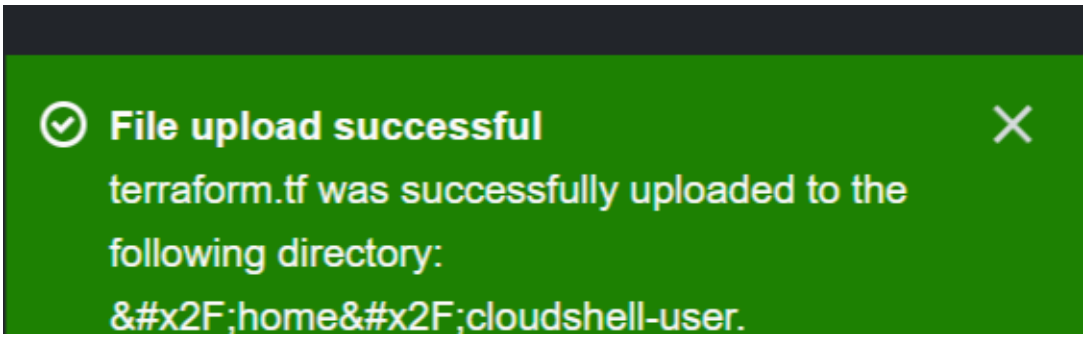
Step 4: Select the terraform file we made “**terraform.tf**”
Step 5: Click “**Open**”



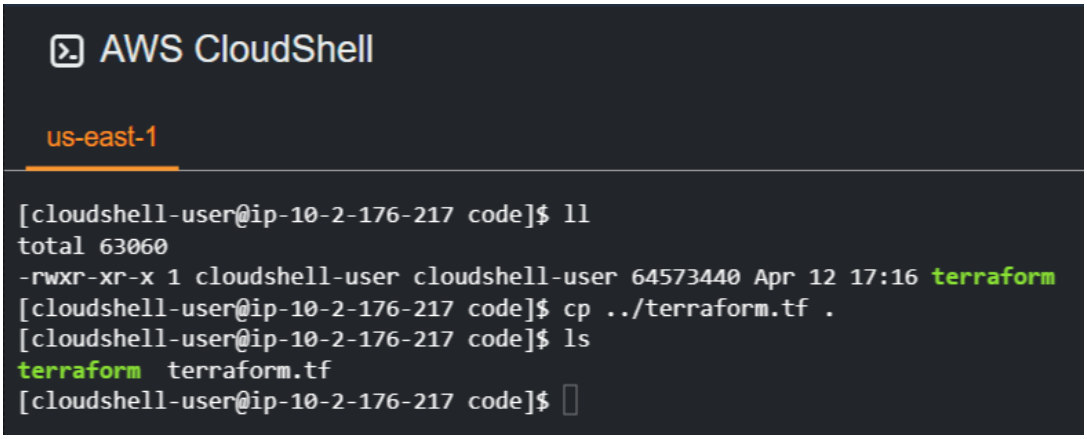
Step 6: Click “**Upload**”



Step 7: Verify Successful Upload



Step 7: The file uploads to the root directory not the current one.
Step 8: Type **”cp ../terraform.tf .”** **>** Hit **”Enter”**



Step 9: Type “**./terraform init**” **>** Hit **”Enter”**

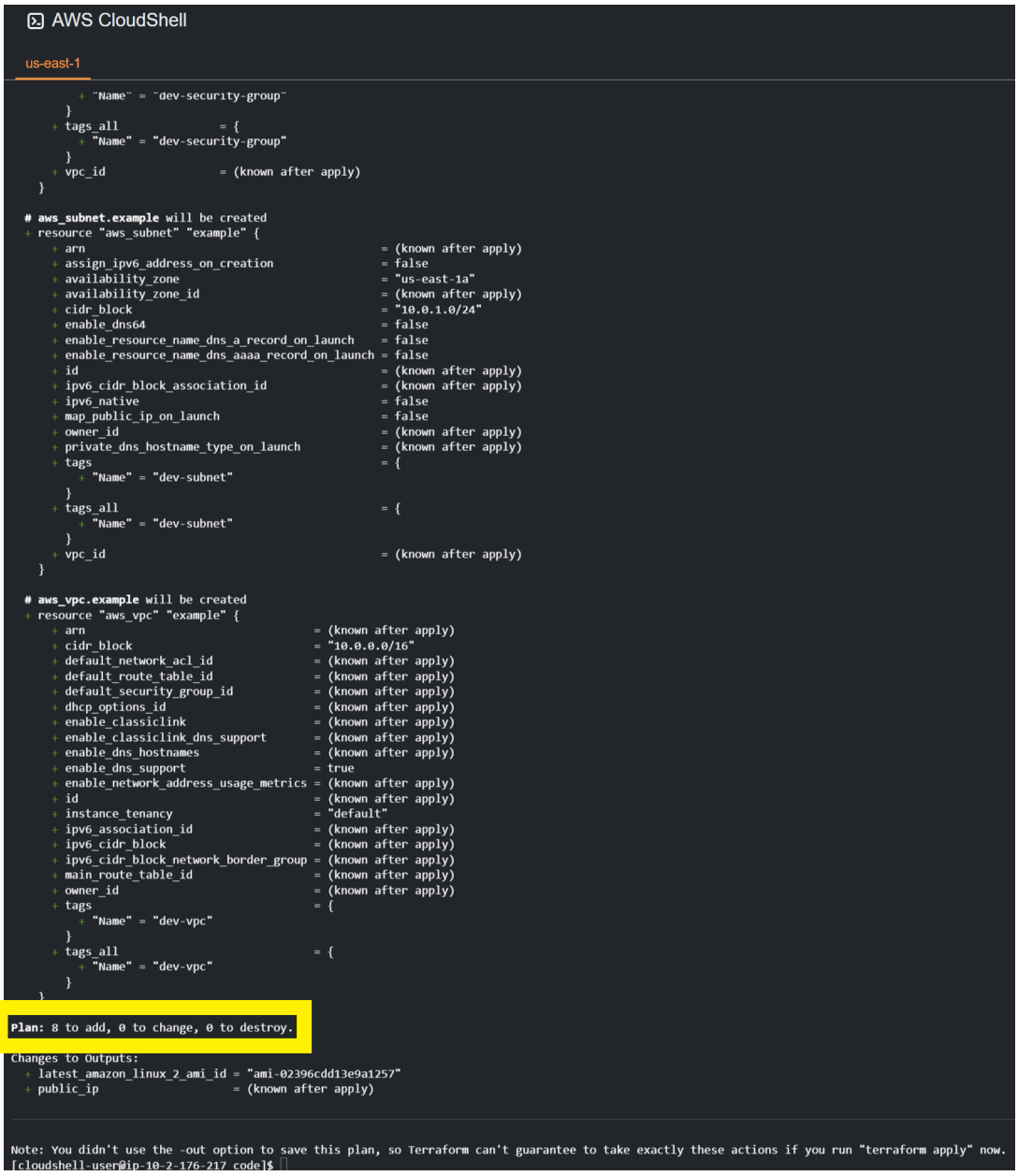
terraform init downloads the AWS provider plugin into the working folder under the directory “.terraform/providers”.

.terraform.lock.hcl contains information about the current state of the dependencies required by the Terraform configuration, including the versions of the providers, modules, and plugins that were used during the last Terraform run.

When Terraform runs, it checks this file to ensure that the dependencies are up-to-date and that there are no conflicts between different versions of the dependencies. If any changes are made to the configuration, Terraform will update the lock file to reflect the new state of the dependencies.

modules (not covered here) allow splitting out groups of Terraform configuration constructs (written in the Terraform language) into reusable abstractions.

Step 10: Type “**./terraform plan**” **>** Hit **”Enter”**



In this image we see that there will be 8 changes made to the existing AWS configuration, Zero changes to be made, and Zero to be destroyed. This is the most important thing to look for when applying your code. This is where you will catch any mistakes you may have made.

Step 11: Type "**./terraform apply**" > Hit "**Enter**"
Step 12: Type "**yes**" > Hit "**Enter**"

AWS CloudShell

us-east-1

```
      + "Name" = "dev-subnet"
    }
  + vpc_id                                     = (known after apply)
}

# aws_vpc.example will be created
+ resource "aws_vpc" "example" {
    + arn                                  = (known after apply)
    + cidr_block                           = "10.0.0.0/16"
    + default_network_acl_id               = (known after apply)
    + default_route_table_id               = (known after apply)
    + default_security_group_id            = (known after apply)
    + dhcp_options_id                      = (known after apply)
    + enable_classiclink                   = (known after apply)
    + enable_classiclink_dns_support       = (known after apply)
    + enable_dns_hostnames                 = (known after apply)
    + enable_dns_support                   = true
    + enable_network_address_usage_metrics = (known after apply)
    + id                                   = (known after apply)
    + instance_tenancy                     = "default"
    + ipv6_association_id                  = (known after apply)
    + ipv6_cidr_block                      = (known after apply)
    + ipv6_cidr_block_network_border_group = (known after apply)
    + main_route_table_id                  = (known after apply)
    + owner_id                             = (known after apply)
    + tags                                 = {
        + "Name" = "dev-vpc"
      }
    + tags_all                             = {
        + "Name" = "dev-vpc"
      }
  }

Plan: 8 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + latest_amazon_linux_2_ami_id = "ami-02396cdd13e9a1257"
  + public_ip                    = (known after apply)

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_vpc.example: Creating...
aws_vpc.example: Creation complete after 1s [id=vpc-00a180d4fc5fcc78e]
aws_subnet.example: Creating...
aws_internet_gateway.example: Creating...
aws_route_table.example: Creating...
aws_security_group.example: Creating...
aws_internet_gateway.example: Creation complete after 0s [id=igw-0beb7065c29086544]
aws_route_table.example: Creation complete after 0s [id=rtb-00e7537368edf4744]
aws_route.default_route: Creating...
aws_subnet.example: Creation complete after 0s [id=subnet-056d86b8c4bc976c6]
aws_route_table_association.example: Creating...
aws_route.default_route: Creation complete after 1s [id=r-rtb-00e7537368edf47441080289494]
aws_route_table_association.example: Creation complete after 1s [id=rtbassoc-04433e312c1fa2e56]
aws_security_group.example: Creation complete after 2s [id=sg-03b18c6ed4a16bea5]
aws_instance.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Creation complete after 12s [id=i-0a59369e98c1b6fc2]

Apply complete! Resources: 8 added, 0 changed, 0 destroyed.

Outputs:

latest_amazon_linux_2_ami_id = "ami-02396cdd13e9a1257"
public_ip = "3.87.70.240"
[cloudshell-user@ip-10-2-176-217 code]$
```

During this final Apply verify the Add, Changes, and Destroy numbers.

Outputs allow you to retrieve information from these created or updated resources, such as IP addresses, resource IDs, or DNS names. Here are two things that are of value. Show the AMI that was chosen and the Public IP address of the server for quick Putty configuration.